



Universidade Federal da Bahia
Instituto de Matemática

Programa de Pós-Graduação em Ciência da Computação

**AN APPROACH FOR RECOVERING
ARCHITECTURAL VARIABILITY FROM
SOURCE CODE**

Crescencio Rodrigues Lima Neto

TESE DE DOUTORADO

Salvador
19 de Fevereiro de 2019

CRESCENCIO RODRIGUES LIMA NETO

**AN APPROACH FOR RECOVERING ARCHITECTURAL
VARIABILITY FROM SOURCE CODE**

Esta Tese de Doutorado foi apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal da Bahia, como requisito parcial para obtenção do grau de Doutor em Ciência da Computação.

Orientadora: Christina Von Flach Garcia Chavez
Co-orientador: Ivan do Carmo Machado

Salvador
19 de Fevereiro de 2019

Ficha catalográfica elaborada pelo Sistema Universitário de Bibliotecas (SIBI/UFBA),
com os dados fornecidos pelo(a) autor(a).

Lima-Neto, Crescencio Rodrigues
An Approach for Recovering Architectural
Variability from Source Code / Crescencio Rodrigues
Lima-Neto. -- Salvador, 2019.
164 f. : il

Orientadora: Christina von Flach Garcia Chavez.
Coorientador: Ivan do Carmo Machado.
Tese (Doutorado - Programa de Pós-graduação em
Ciência da Computação) -- Universidade Federal da
Bahia, Instituto de Matemática e Estatística, 2019.

1. Software Product Lines. 2. Product Line
Architecture. 3. Software Architecture. 4.
Variability. 5. Software Architecture Recovery. I.
Chavez, Christina von Flach Garcia. II. Machado, Ivan
do Carmo. III. Título.

CRESCENCIO RODRIGUES LIMA NETO

**“AN APPROACH FOR RECOVERING ARCHITECTURAL
VARIABILITY FROM SOURCE CODE”**

Esta tese foi julgada adequada à
obtenção do título de Doutor em Ciência
da Computação e aprovada em sua
forma final pelo Programa de Pós-
Graduação em Ciência da Computação
da UFBA.

Salvador, 19 de fevereiro de 2019.



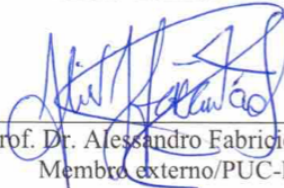
Prof. Dr. Ivan do Carmo Machado
Co-orientador/PGCOMP



Prof. Dr. Cláudio Nogueira Sant'Anna
Membro interno/PGCOMP



Prof. Dr. Manoel Gomes de Mendonça Neto
Membro interno/PGCOMP



Prof. Dr. Alessandro Fabricio Garcia
Membro externo/PUC-Rio



Prof.ª Dr.ª Thelma Elita Colanzi
Membro externo/ DIN-UEM

To my family...

ACKNOWLEDGEMENTS

First, I would like to thank my family. Especially to my father and mother, without their guidance I would never get this far; my brother and sister for the constant support; my wife and kids for their understanding. I love them and appreciate the efforts they have put into giving me the conditions to finish this work.

I would like to grateful acknowledge my advisors, Christina and Ivan. They provided me with many helpful suggestions and encouragement during this work as well as the challenging research that lies behind it.

The results of this thesis could not be achieved without the support of the aSide Labs and Reuse in Software Engineering (RiSE) Labs. My gratitude to all my friends from the research labs. I also thank my friends and colleagues that I have met during my journey in Salvador since 2011.

There were a lot of people that directly or indirectly participated with me along my Ph.D. path. However, it will not be possible to remember of all of them now. My excuses for the ones I forgot. My thanks are extended for the Fundação de Amparo à Pesquisa do Estado da Bahia (Fapesb) and Federal Institute of Bahia (IFBA) for partially supporting this research project.

Finally, I would like to thank God for giving me the wisdom to perform this work.

*Try not to become a man of success, but rather
try to become a man of value.*

—ALBERT EINSTEIN

RESUMO

Engenharia de Linha de Produto de Software (SPLE) tem sido amplamente adaptada para aplicação de reuso sistemático em famílias de sistemas. Devido ao alto investimento prévio necessário para adoção de SPLE, organizações geralmente começam com abordagens de reuso oportunistas (e.g., sistemas que são clonados e modificados). No entanto, problemas de manutenção aparecem ao gerenciar um grande número de sistemas semelhantes onde cada um implementa e evolui características particulares. Uma solução viável para resolver esse problema é migrar para Linhas de Produto de Software (SPLs) usando uma abordagem extrativa. Essa iniciativa, em suas fases iniciais, inclui a definição de uma Arquitetura de Linha de Produtos (PLA) para apoiar a derivação de produtos variantes e também permitir a customização de acordo com as necessidades dos clientes. Desta forma, o uso sistemático de técnicas de Recuperação de Arquitetura de Software (SAR) permitem a recuperação da PLA e mantêm a PLA alinhada com o desenvolvimento. Nosso objetivo é fornecer uma abordagem para recuperar PLAs e diretivas para apoiar a recuperação das PLAs. Nós reunimos conhecimento por meio de revisões da literatura e estudos exploratórios para caracterizar o estado-da-arte e identificar oportunidades de pesquisa em técnicas e ferramentas de SAR que apoiam a recuperação de variabilidade arquitetural a partir de informações provenientes do código fonte para uma família de produtos. O uso de técnicas e ferramentas de SAR para recuperar uma PLA que documenta informação sobre variabilidade no nível arquitetural pode atacar problemas relacionados com a adoção, design, e evolução de SPLs. Infelizmente, poucos estudos investigam a recuperação de PLA e também fornecem avaliação empírica. Um dos principais problemas da abordagem extrativa é a explosão da variabilidade na representação arquitetural. Nossa abordagem é baseada na identificação da variabilidade em nível arquitetural por meio da extração de informações do código fonte dos variantes. Para avaliar nossa abordagem, realizamos um conjunto de estudos empíricos. Os resultados forneceram evidências de que nossa abordagem é capaz de recuperar PLAs, identificar a variabilidade a nível arquitetural e filtrar variantes exclusivos através da eliminação de pacotes e classes específicas sem precisar remover os variantes completamente. Nós consideramos que a PLA recuperada pode ajudar especialistas do domínio na tomada de decisão para apoiar a adoção de SPLE.

Palavras-chave: Linha de Produto de Software, Arquitetura de Linha de Produto de Software, Recuperação de Arquitetura de Software.

ABSTRACT

Software Product Line Engineering (SPLE) has been widely adopted for applying systematic reuse in families of systems. Given the high upfront investment required for SPLE adoption, organizations commonly start with more opportunistic reuse approaches (e.g., a single system that they clone and modify). However, maintenance problems appear when managing a large number of similar systems where each of them implements and evolves particular characteristics. One viable solution to solve this issue is to migrate to SPLs using an extractive approach. This initiative, in its early phases, includes the definition of a Product Line Architecture (PLA) to support the derivation of product variants and also to allow customization according to customers' needs. In this way, the systematic use of Software Architecture Recovery (SAR) techniques enables PLA recovery and keeps the PLA aligned with development. Our objective is to provide an approach to recover PLAs and guidelines to support the PLA recovery. We gathered knowledge by means of literature reviews and exploratory studies to characterize the state-of-the-art and identify research gaps on SAR techniques and tools that support the recovery of architectural variability information from source code for a family of products. The use of SAR techniques and tools to recover a PLA that documents variability information at the architecture level may address issues related to SPL adoption, design and evolution. Unfortunately, few studies investigate PLA recovery and also provide empirical evaluation. One of the main issues in the extractive approach is the explosion of the variability in the PLA representation. Our approach is based on identifying variability on architectural level by extracting information from variants' source code. To evaluate our approach, we performed a set of empirical studies. The results provided evidence that our approach is able to recover PLAs, identify the variability at architectural level, and filter outliers variants, allowing the elimination of 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, Software Architecture Recovery.

CONTENTS

List of Figures	xix
List of Tables	xxii
List of Acronyms	xxiv
Chapter 1—Introduction	1
1.1 Problem statement	2
1.2 Objectives	3
1.3 Research Questions	3
1.4 Research Methods	4
1.4.1 Part 1. Literature Review	4
1.4.2 Part 2. Concept	4
1.4.3 Part 3. Empirical Studies	5
1.5 Contributions of this Thesis	5
1.6 Out of Scope	6
1.7 Thesis Outline	6
Chapter 2—Background	9
2.1 Software Product Lines	9
2.1.1 Software Product Line Engineering	12
2.1.2 Variability Management	14
2.2 Software Architecture	16
2.2.1 Architecture Descriptions	17
2.2.2 Architecture Variability	20
2.3 Product Line Architecture	22
2.4 Software Architecture Recovery	24
2.4.1 Processes	25
2.4.2 Techniques	26
2.4.3 Tools	27
2.4.4 Product Line Architecture Recovery	28
2.5 Reuse Assessment	28
2.6 Chapter Summary	30

Chapter 3—A Systematic Mapping Study on Product Line Architecture Recovery	33
3.1 Motivation	33
3.2 Research Process	34
3.2.1 Research Questions (RQs)	34
3.2.2 Search Strategy	35
3.2.3 Selection Criteria	35
3.2.4 Data Sources	36
3.2.5 Data Collection	36
3.2.6 Data Analysis	36
3.2.7 Quality Assessment	38
3.3 Outcomes	38
3.3.1 Characteristics of the studies	39
3.3.2 Results	40
3.3.3 RQ 1: How does the relationship between PLA and SAR evolve over the years?	41
3.3.4 RQ 2: How does the existing solution proposal support PLA recovery?	42
3.3.5 RQ 3: What are the PLA recovery trends?	44
3.4 Discussion	45
3.4.1 Main Findings	45
3.4.2 Limitations of the Review and Threats to Validity	51
3.5 Related Work	51
3.6 Chapter Summary	52
Chapter 4—An Approach for Recovering PLA from Source Code of Variants	53
4.1 Overview	53
4.1.1 Purpose of the PLA recovery	55
4.1.2 Illustrative Example	55
4.2 A Metamodel for PLA Description	58
4.3 PLA Recovery	58
4.3.1 Techniques	58
4.3.2 Activities	62
4.3.3 Supporting Tools	65
4.4 Guidelines for PLA Recovery	67
4.4.1 Clone-and-Own	67
4.4.2 Generate Variants	70
4.4.3 Analyze #ifdefs	72
4.5 Related work	74
4.6 Chapter Summary	75

Chapter 5—Exploratory Study on PLA Recovery	77
5.1 Design	77
5.1.1 Planning	77
5.1.2 Study Design	79
5.1.3 Exploratory Study materials	80
5.1.4 Subjects	81
5.1.5 The Study Projects	81
5.2 Execution	81
5.2.1 Procedure	82
5.3 Analysis and Interpretation	82
5.3.1 RQ 1: Does SAVaR provide a precise and reliable version of the implemented PLA?	83
5.3.2 RQ 2: How much detail is needed to represent the recovered PLA?	83
5.3.3 RQ 3: Do the metamodels (for PLA design) support on the under- standing of the recovered PLA?	83
5.3.4 Feedback	84
5.3.5 Metrics Analysis - Descriptive Statistics	85
5.4 Discussion	87
5.4.1 Main Findings	87
5.4.2 Variability Identification	88
5.4.3 Amount of details in the PLA recovery	88
5.4.4 Filtering the recovered PLA	89
5.4.5 Analyzing the recovered PLA based on Metamodels	90
5.4.6 Analyzing the changes from SAR to PLA recovery	91
5.4.7 Threats to Validity	91
5.5 Related Work	92
5.6 Chapter Summary	92
Chapter 6—Recovering the PLA of 15 open source SPL projects	95
6.1 Study Design	95
6.1.1 Research Questions	95
6.1.2 Hypotheses	96
6.1.3 Metrics	96
6.1.4 Analysis Procedure	97
6.2 Study Operation	97
6.2.1 SPL Projects analyzed	97
6.2.2 Preparation	98
6.2.3 Data collection	99
6.3 Data Analysis	99
6.3.1 Descriptive Statistics	99
6.3.2 Draw Product Line Results	100
6.3.3 Video on Demand Results	101
6.3.4 Zip Me Results	101

6.3.5	Game of Life Results	101
6.3.6	Graph Product Line Results	102
6.3.7	Prop4J Results	102
6.3.8	BankAccount and BankAccountV2 Results	103
6.3.9	DesktopSearcher Results	104
6.3.10	PayCard Results	104
6.3.11	PokerSPL Results	104
6.4	Discussion	104
6.4.1	Answers to the Research Questions	104
6.4.2	General Findings	106
6.4.3	Threats to Validity	107
6.5	Chapter Summary	107
Chapter 7—Case Studies		111
7.1	Recovering the PLA of the Apo-Games	112
7.1.1	Case Study	113
7.1.2	Threshold Configurations	114
7.1.3	Results	114
7.1.4	Discussion	117
7.1.5	RQ1 - How similar the variants are	117
7.1.6	RQ2 - Correlation between variants' size and likely outliers	119
7.1.7	RQ3 - Impact of outliers removal in the recovery of better PLAs	119
7.1.8	Threats to Validity	119
7.2	Recovering the PLA of 10 open source projects	120
7.2.1	Preparation	122
7.2.2	Analysis and Interpretation	122
7.2.3	Threats to Validity	123
7.3	Chapter Summary	124
Chapter 8—Conclusion		129
8.1	Related work	129
8.2	Contributions	130
8.3	Research Limitations	130
8.4	Future Research Directions	131
8.5	Contributions so far	132
Appendix A—Mapping Study on PLA Recovery - Primary Studies and data set		147
A.1	List of Journals	147
A.2	List of Conferences	148
A.3	Primary studies	149

Appendix B—Recovering PLA from Variants’ Source Code – Additional Material151

 B.1 Motivating example additional material 151

 B.1.1 MobileMedia Variants’ extracted architecture 151

 B.1.2 MobileMedia SPL recovered PLA 151

Appendix C—Exploratory Study on Product Line Architecture Recovery - Data set 157

 C.1 Consent form 157

 C.2 Background Questionnaire 157

LIST OF FIGURES

1.1	Thesis Outline	7
2.1	Conceptual elements of SPL and SA	10
2.2	Feature Model for MobileMedia.	11
2.3	Core Asset Development	13
2.4	Product Development	14
2.5	Excerpt from ISO/IEC/IEEE 42010 and design elements	17
2.6	The 4+1 view model	19
2.7	DSM for MobileMedia	20
2.8	MDG for MobileMedia	21
2.9	Excerpt from Feature Variability Extension	22
2.10	Variability Model for MobileMedia	23
2.11	A bottom-up process	25
2.12	A top-down process	26
3.1	Paper selection flowchart	37
3.2	Temporal view of the studies	39
3.3	Timeline	41
3.4	Research types per research contribution	43
3.5	Number of studies per SAR taxonomy axes	44
3.6	Contribution type versus research type versus research questions	46
3.7	Evolution lines identified	47
3.8	Empirical studies over the years	47
3.9	Research type over the years	48
3.10	SAR coals vs. research type vs. empirical	49
3.11	SAR processes over the years	50
3.12	SAR goals vs. SAR processes vs. SAR techniques	50
4.1	PLA Recovery Main Phase	54
4.2	Operation phase in details	54
4.3	Extracted information from three Variants (packages and relations)	56
4.4	Recovered PLA – development view	57
4.5	Recovered DSM of the PLA	58
4.6	Conceptual elements of SPL and SA	59
4.7	PLA recovery based on variants’ architecture	60
4.8	PLA recovery based on <code>#ifdef</code> directives.	61
4.9	Application of the threshold in the MobileMedia (classes)	63

4.10	PLA Recovery Details	65
4.11	Sugestion of tool chain for Clone-and-Own guideline	70
4.12	Sugestion of tool chain for Generate Variants guideline	72
4.13	Sugestion of tool chain for Analyze #ifdefs guideline	74
5.1	Study Design Phases	80
5.2	Research Question 1 Answers	83
5.3	Research Question 2 Answers	84
5.4	Research Question 3 Answers	84
5.5	Approach feedback part 1	85
5.6	Approach feedback part 2	85
5.7	Boxplot – Component Reuse Rate per group	86
5.8	Boxplot – Relation Reuse Rate per group	87
5.9	Recovered Information isolated by the feature Comment	89
5.10	Example of class with all the recovered information	90
6.1	The overall recovery process: activities, inputs and outputs.	98
6.2	Boxplot of Component Reuse Rate per PLA	100
6.3	Design Structure Matrix for GOL	102
6.4	Design Structure Matrix for Prop4J	103
6.5	Comparing SSC and SVC metrics	108
6.6	Correlation analysis	109
7.1	Variants’ architectural similarity graph	114
7.2	Concept lattice with variants in the games	115
7.3	LOC per Exclusive Variants - Packages (cor. 0.51)	116
7.4	LOC per Exclusive Variants - Classes (cor. 0.80)	116
B.1	Extracted information from MobileMedia Variant 4	151
B.2	Extracted information from MobileMedia Variant 5	152
B.3	Extracted information from MobileMedia Variant 6	152
B.4	Extracted information from MobileMedia Variant 7	153
B.5	Extracted information from MobileMedia Variant 8	153
C.1	Recovered Design Structure Matrix - Project SPL Web Store	160
C.2	Recovered PLA (module view) - Project SPL Web Store	161
C.3	Recovered Design Structure Matrix - Project SPL Message	162
C.4	Recovered PLA (module view) - Project SPL Message	163

LIST OF TABLES

2.1	Metrics used to evaluate the PLA.	29
2.2	Recovered Metrics for MobileMedia	30
2.3	CRR for MobileMedia	30
2.4	RRR for MobileMedia	31
3.1	Study Quality Assessment Criteria	38
3.2	Classification of Reviewed Studies	40
3.3	Number of studies per research type	42
3.4	Solution proposal per research contribution	43
3.5	Number of studies per research question	51
4.1	MobileMedia variants' description	55
4.2	Guidelines for PLA Recovery.	68
5.1	SPL Projects – Metrics	82
5.2	Study Execution Agenda	82
5.3	Recovered Metrics from the PLAs	86
6.1	SPL Projects analyzed and Metrics collected for PLAs	99
6.2	CRR Measures for DPL elements	100
6.3	CRR Measures for Prop4J	105
6.4	Comparisons that rejected the null hypothesis RQ1	105
6.5	Comparisons that rejected the null hypothesis RQ2	106
7.1	Describing the study according to GQM	113
7.2	Apo-Games Projects – Metrics summary	113
7.3	Recovered Metrics from the PLAs (Packages)	117
7.4	Recovered Metrics from the PLAs (Classes)	118
7.5	PLA Metrics after eliminating some variants and Threshold analysis	118
7.6	Describing the study according to GQM	121
7.7	Analyzed Projects	122
7.8	Recovered Metrics from the PLAs	126
8.1	SPL Projects	133
A.1	List of Journals	147
A.2	List of Conferences	148
A.3	Selected primary studies	149

C.1 Consent Form 158

C.2 Characterization form used 159

C.3 Subjects' Profile 164

C.4 Subjects Experience with Programming Language 164

LIST OF ACRONYMS

ADL	Architecture Description Language
CAD	Core Asset Development
DSM	Design Structure Matrix
FCA	Formal Concept Analysis
IDE	Integrated Development Environment
GQM	Goal Question Metric
GUI	Graphical User Interface
PD	Product Development
PLA	Product Line Architecture
PLAR	Product Line Architecture Recovery
SA	Software Architecture
SAR	Software Architecture Recovery
SAVaR	Software Architectre Variability Recovery
SEI	Software Engineering Institute
SLR	Systematic Literature Review
SMS	Systematic Mapping Study
SPEM	Software and Systems Process Engineering Meta-model
SPL	Software Product Line
SPLA	Software Product Line Architecture
SPLE	Software Product Line Engineering
UFBA	Federal University of Bahia
UML	Unified Modeling Language
XML	eXtensible Markup Language

Chapter

1

I have no special talents, I am only passionately curious – Albert Einstein

INTRODUCTION

Variability is a relevant characteristic of software systems (HILLIARD, 2010; GALSTER; AVGERIOU, 2011a). Supporting variability is essential to manage commonalities and differences across software, and to accommodate reuse in different organizations and product versions (GALSTER; AVGERIOU, 2011b). Ideally, variability should be identified and managed early in the life cycle (THIEL; HEIN, 2002a). Moreover, since variability can be a complex and multi-faceted concept, it should be treated as a first-class citizen in software architecture documentation (GALSTER et al., 2013).

In Software Product Lines (SPL), variability is often handled during feature modeling and product configuration (APEL et al., 2013). At the software architecture level, variability may be supported by a Product Line Architecture (PLA), a core architecture for every SPL product that documents the variation points and variants defined in the SPL variability model (POHL; BÖCKLE; LINDEN, 2005). The PLA is expected to provide high-level descriptions of mandatory, optional, and variable components in the SPL, and their interconnections (GOMAA, 2004). When explicitly documented, the PLA can be useful in different contexts that require variability management. It enables companies to amortize the effort of software design and development over multiple products, thereby reducing costs (BOSCH, 1999).

For instance, legacy software systems commonly have not been created with an SPL perspective (ANGERER et al., 2014). It is not uncommon for small and medium-sized companies to slowly migrate legacy software systems to SPLs and adopt it using a *clone-and-own* approach, by copying, adding or removing functions from existing products. This approach leads to *ad-hoc* product portfolios of multiple yet similar variants (RUBIN; CHECHIK, 2012; FISCHER et al., 2014). With the growth of products portfolio, the management of variability and reuse becomes more complex. In this context, a PLA for the candidate SPL could be *recovered* from existing variants with the support of software architecture recovery techniques.

SPL evolution can also benefit from a documented PLA. In this case, an existing SPL with a feature model and automatic product generation, may be subject to the addition

of new features or the removal of deprecated ones. In this context, a PLA could be *recovered* from existing variants to support development activities such as improving the SPL source code.

Software Architecture Recovery (SAR) aims at providing solutions to problems related to the absence of a documented software architecture (DUCASSE; POLLET, 2009). A software architecture can be recovered from source code and other available information sources (DUCASSE; POLLET, 2009; CLEMENTS et al., 2010). In the context of SPL, PLA recovery requires the analysis of several product variants to support variability identification and then variability documentation at the architecture level.

The need for variability identification from a (possibly) large number of product variants that implement common and variable features poses challenges to the PLA recovery process. First, variability leads to a potentially large configuration space, that grows exponentially based on the number of configuration options that can be set, which in turn makes variability identification a computationally expensive task. Also, variability identification at architectural level will demand additional effort from architects and engineers. The variability representation may include potential noise in the common structure, require specific analysis, and hinder overall understanding. Software engineers developing or maintaining products based on a PLA require considerable knowledge of the rationale and concepts underlying the SPL and the concrete structure of the reusable assets that are part of the PLA (BOSCH, 1999). Moreover, some product variants may become highly unrelated to their predecessors and become outliers (WILLE et al., 2018). The inclusion of outlier variants in PLA recovery may increase the effort for variability identification. Finally, different product implementation strategies may require different recovery techniques.

SAR in the context of SPL is yet an underexplored research topic (LIMA-NETO et al., 2015). Existing work on reengineering product variants into an SPL is mostly focused on variability management at the requirements level (SHATNAWI; SERIAI; SAHRAOUI, 2016). Moreover, existing research on PLA recovery neither addresses the aforementioned challenges nor provides sound empirical evaluation or detailed information to guide recovery and support study replication (SINKALA; BLOM; HEROLD, 2018).

1.1 PROBLEM STATEMENT

There is a lack of SAR approaches that provide architectural variability models to support variability management at the software architecture level (KOSCHKE et al., 2009; LIMA-NETO et al., 2015; SHATNAWI; SERIAI; SAHRAOUI, 2016). These architectural models provide an explicit bridge between variability represented in feature models and variability implemented in product variants. For SPLs, the PLA is expected to play such role. Therefore, the main problem investigated in the scope of this thesis is:

Existing software architecture recovery approaches lack adequate support for recovering product line architectures.

By “lack of adequate support” we mean that existing bottom-up approaches do not address or address partially the recovery of variability information and its representation at the software architecture level that take into account:

- The different implementation of variability mechanisms that may generate or end up with variants' source code that may have different characteristics (for instance, the presence of `#ifdef` annotations);
- The influence of the size of configuration options on the recovering process and strategies to tame it;
- The need for guidelines to ease architecture recovery for SPL projects;
- The need of empirical studies to validate the recovery process and its outputs, under different perspectives.

1.2 OBJECTIVES

The main objective of this thesis is *to investigate and provide adequate support for recovering product line architectures* to be used in the context of variability management at the software architecture level. This objective is decomposed and reified in the following specific objectives:

1. Investigate the state-of-the-art on bottom-up PLA recovery;
2. Propose a systematic approach for recovering PLAs using bottom-up process, techniques, and tool support;
3. Evaluate **SAVaR** by recovering the PLA of real-world projects.

1.3 RESEARCH QUESTIONS

Based on the research objectives, we define the main research question that drives this investigation:

How to achieve variability-aware PLA recovery in a systematic way?

The following research questions address problems related to PLA recovery:

RQ1 How do different variability implementation mechanisms affect PLA recovery?

With this question we aim to understand variability implementation mechanisms and investigate the need for different variability-aware recovery techniques. For instance, how do existing (bottom-up) SAR techniques deal with annotation-based variability in the variants' source code? And how do such techniques deal with clone-and-own variants' source code? Chapter 5, Chapter 6, and Chapter 7 focused on answering this research question.

RQ2 How can we tame the influence of the size of configuration options on the variability identification efficiency?

With this question we investigate the impact of the SPL variability spectrum in a PLA recovery process based on product variants. We analyze how the integration of

outlier variants demand additional effort for identifying the mandatory and variable elements of the recovered PLA. We performed studies on Chapter 6 and Chapter 7 to answer this research question.

Furthermore, we are concerned with the soundness of PLA recovery processes and with the accuracy and understandability of the recovered PLAs. Therefore, we aim to address the following research question as well:

RQ3 Are the PLAs recovered from variants accurate and understandable?

With this question, we investigate if the recovered PLA is accurate with respect to variability implemented in product variants and if variability-aware architecture models are easy to understand in SPL adoption and evolution scenarios. Chapter 5 presents the empirical study to answer this research question.

1.4 RESEARCH METHODS

Based on the research objectives and questions, we applied a combination of methods, to gain an in-depth understanding of the research problem and to strengthen our study conclusions (HESSE-BIBER, 2010).

1.4.1 Part 1. Literature Review

The initial part of the thesis comprises the analysis of existing literature that supports this investigation, as a means of devising our research questions, and narrowing down the possibilities to consider. Moreover, it guided the early steps of our research.

No systematic literature reviews that investigated software architecture recovery in the context of SPL were found. Therefore, we performed a systematic mapping study (SMS) (LIMA-NETO et al., 2015) to gather data and evidence from primary studies in which these two fields meet. Moreover, we performed a systematic literature review (SLR) to synthesize evidence about metamodels used by PLA modeling, design or recovery approaches (LIMA; CHAVEZ, 2016).

Some findings from these studies are the lack of studies that address PLA recovery and the lack of guidelines to support the PLA recovery. Moreover, most of the existing metamodels to design PLA considered the variability representation using variation points.

1.4.2 Part 2. Concept

The second part describes the core of this research.

First, we performed an exploratory study to analyse the feasibility of adapting existing SAR tools and techniques to support the design of PLAs (LIMA-NETO et al., 2015). We recovered the architecture of the SPL variants to create the PLA. Furthermore, we developed a semiautomatic approach to support PLA recovery from the source code of *clone-and-own* SPL products.

Then, we proposed a Software Architecture Variability Recovery (SAVaR), an approach to PLA recovery with the goal of providing variability-aware software architecture

documentation for SPLs, while addressing the problems presented in Section 1.1. **SAVaR** supports variability identification based on the source code of the variants. We developed tool support to extract, identify and document architecture variability information for SPL from products implemented with different strategies. Variability traceability is supported by means of a single metamodel used to generate architectural views. Finally, a set of documented guidelines supports **SAVaR** and provide guidance to developers and architects.

1.4.3 Part 3. Empirical Studies

We performed a set of empirical studies to evaluate **SAVaR**. First, we performed an exploratory study to analyze how to recover PLAs by applying **SAVaR** with the purpose of understanding with respect to its effectiveness and reliability from the point of view of developers and recoverers in the context of SPL projects in an academic environment (Chapter 5). This exploratory study helped us to improve **SAVaR** and related guidelines.

Secondly, we performed studies with several open-source SPL projects to assess the quality of the PLAs recovered with the support of **SAVaR** (Chapter 6). The objective of the second study was to analyze a PLA with the purpose of understanding how variability affects architecture recovery with the PLAR tool from the point of view of architects performing variability identification in the context of SPL projects.

Finally, the objective of the last study was to evaluate whether **SAVaR** supports a cost-effective PLA recovery by means of the identification and removal of outliers. We performed two studies: the first one focused on recovering the PLA for the Apo-Games project in the context of variants developed using clone-and-own strategy and the second study focused on recovering the PLAs of 10 open source SPL projects (Chapter 7).

1.5 CONTRIBUTIONS OF THIS THESIS

This thesis focuses on variability-aware software architecture recovery for SPL. We address PLA recovery to provide high-level architectural models for the variability implemented in SPL products, and therefore extend variability management with variability-aware architectural assets that can be used in the context of SPL adoption and evolution.

The main contributions of this thesis include:

1. Summary of the state-of-the-art on PLA, with challenges and primary studies that address different aspects of the research area;
2. Analysis of gaps, challenges, insights, and problems faced by SPL architects and developers during PLA design;
3. **SAVaR**, an approach that supports PLA recovery, with variability identification and filtering of outlier variants. Threshold analysis, outliers filtering and metrics-based analysis were combined to improve the recovery process and the recovered PLAs.
4. A set of guidelines to document and systematize the **SAVaR** recovery process.

Finally, we performed a set of empirical studies with different open source SPLs to evaluate the proposed approach. These studies contribute to evidence-based software engineering by providing data and procedures that may support replication by other researchers interested in the field of PLA recovery.

1.6 OUT OF SCOPE

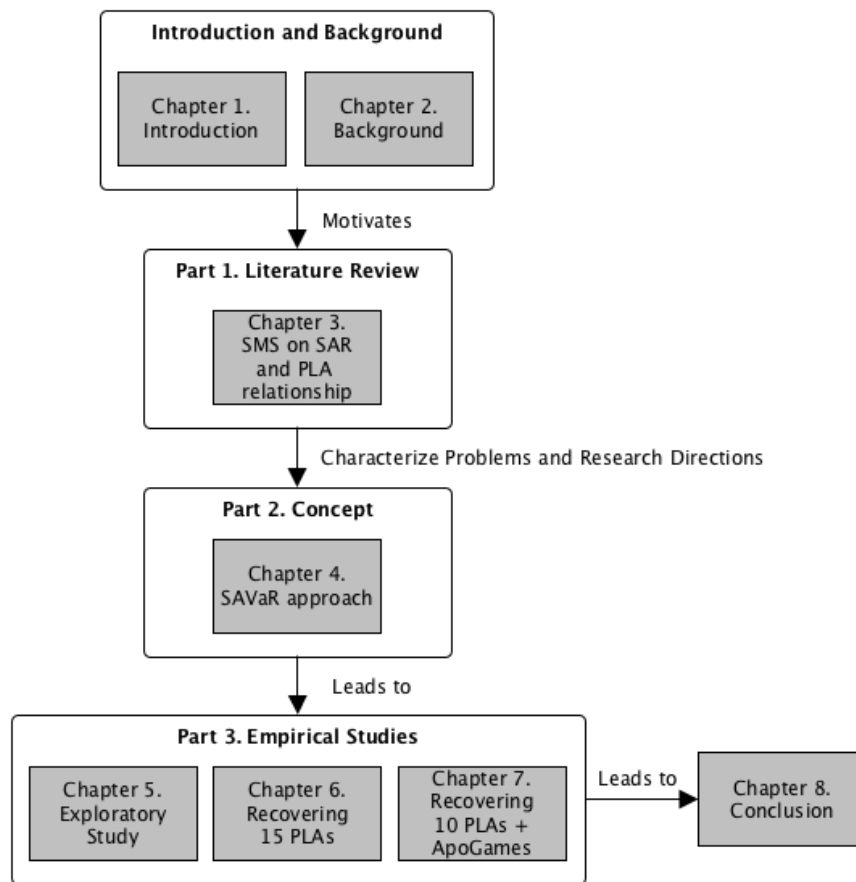
The following topics are out of the scope of this thesis:

- The concept of reference architectures. There seems to be a misconception in some related work on what is a PLA and what is a reference architecture. We focus our study on the former because we investigated the SPL variants source code;
- Hybrid and top-down software architecture reconstruction processes. Nevertheless, they were included in the SMS to cover the research in the SAR field and identify possible gaps. In this thesis, we focus on bottom-up processes.
- The use of dynamic analysis techniques and models. We focus on static analysis. For this reason, extracted information during runtime was not considered.

1.7 THESIS OUTLINE

Figure 1.1 presents the thesis outline. Chapter 2 introduces the necessary background for this thesis. Chapter 3 presents the SMS conducted in this thesis. We discuss the relationship between SAR and PLA that leads to important findings in the research area. Chapter 4 describes the **SAVaR** approach and the guidelines developed to support the PLA recovery. Chapter 5, 6, and 7 evaluate the **SAVaR** approach and discuss the findings and suggestions for improvement. Finally, Chapter 8 concludes the thesis and discusses future work.

Appendix A presents supporting material for the SMS, Appendix B provides complementary information about **SAVaR** and Appendix C provides supporting material for Chapter 5 that includes information about subjects and questionnaires.

**Figure 1.1** Thesis Outline

Don't complain; just work harder. – Randy Pausch

BACKGROUND

This chapter presents the conceptual background that supports our research. Figure 2.1 is used to guide the presentation of background on SPL (Section 2.1), software architecture (Section 2.2), product line architecture (Section 2.3), and software architecture recovery (Section 2.4). Terms and definitions introduced along the text in italics, e.g. (*Concept Name*) refer to the conceptual elements presented in the figure. Other important background on reuse assessment is presented in Section 2.5. Background on PLA is presented in Chapter 3.

2.1 SOFTWARE PRODUCT LINES

A Software Product Line (SPL) is a set of software products offered by a producer to customers, which provides the production for a mass market cheaply than individual product creation (POHL; BÖCKLE; LINDEN, 2005).

Clements and Northrop (2001) define a SPL as “a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission developed from a common set of core assets in a prescribed way”. A “core asset” is anything used to produce multiple products (source code, software architecture, test infrastructure, test cases, test data, production plans, etc). The assets are designed to handle the range of variability defined in the SPL scope, and each one is accompanied by an attached process.

Apel et al. (2013) state that “instead of developing software systems from scratch, they should be constructed from reusable parts. Instead of composing software systems always in the same way, they should be tailored to requirements of the customer, where customers can select from a large space of configuration options.”

SPL products share a set of commonalities and have variabilities that make them unique (POHL; BÖCKLE; LINDEN, 2005). A *commonality* is a characteristic that is common to all products. On the other hand, a *variability* is a characteristic that may be present in some products, but not in all (LINDEN; SCHMID; ROMMES, 2007). Commonalities and variabilities are described in terms of features (GOMAA, 2004).

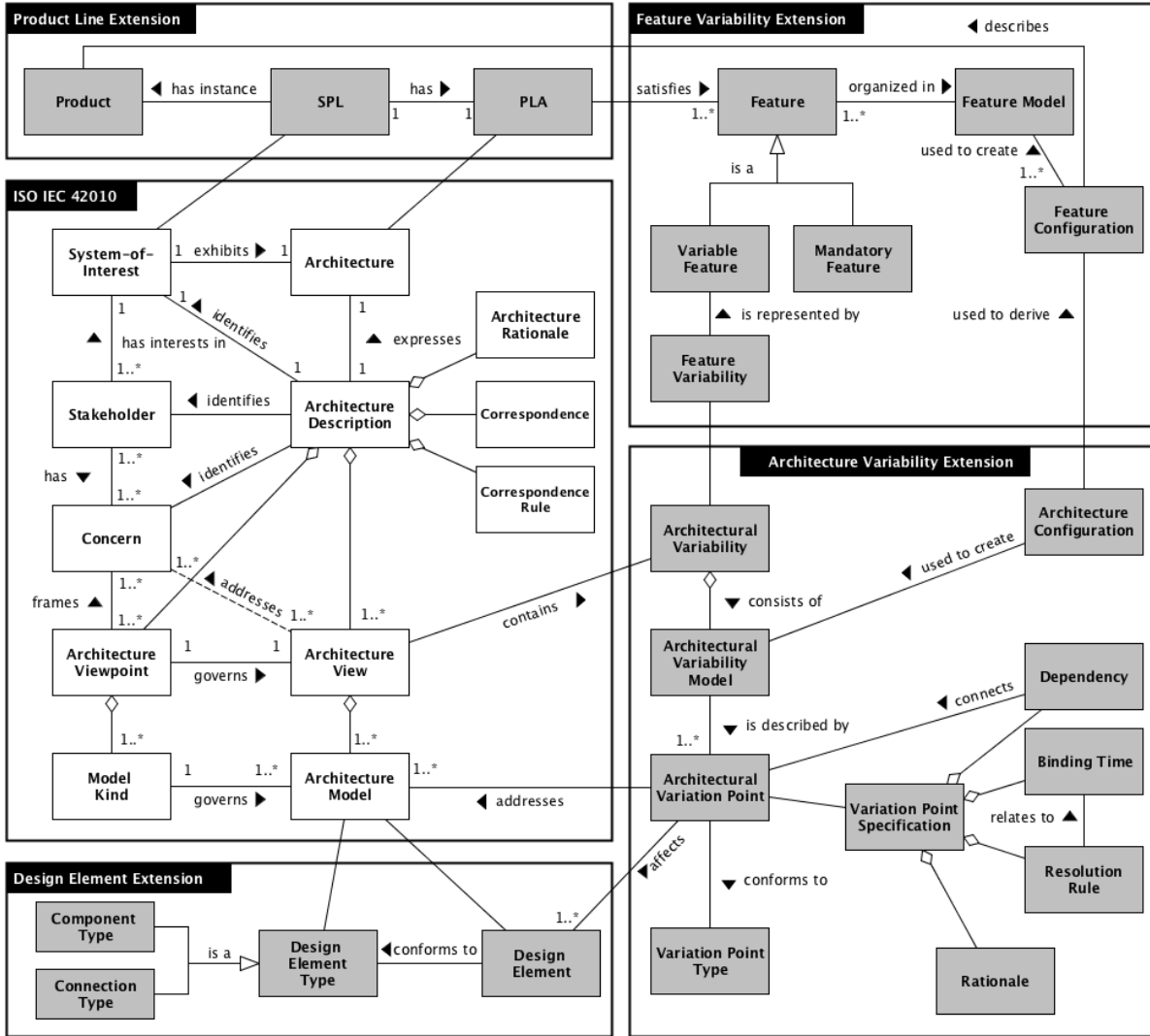


Figure 2.1 Conceptual elements of Software Product Lines and Software Architecture – adapted from (THIEL; HEIN, 2002b)

A *Feature*q is a “prominent or distinctive user-visible aspect, quality, or characteristic of a software system or systems” (KANG et al., 1990). Each SPL feature can be mandatory (*Mandatory Feature*) or variable (*Variable Feature*). Mandatory features must be present in all SPL products. On the other hand, SPL products can differ from each other by their variable features; such features are ‘variable’ assets that will be present only in some products. A variable feature can be categorized as an *optional* or *alternative* feature. Optional features are present in only some SPL products. When two or more features are alternatives to each other, only one of them can be present in a given product. *Feature Variability* is represented by variable features. In the context of our research, we are interested in *Architectural Variability* and how it addresses feature variability.

Features can be organized by means of tree-like structures known as feature models (KANG et al., 1990). A *Feature Model* represents variability and describes different

types of features, their relationships and dependencies. Each feature in the model may have a set of child features with a given type of relationship: mandatory, optional, or alternative. They describe the features that can appear in a member of the SPL, to separate the mandatory features from the variable ones, and to indicate how the variable features can appear (CZARNECKI; EISENECKER, 2000).

Feature models are used to create *Feature Configurations* that in turn are used to derive *Architecture Configurations* which describes the components necessary to compose the architecture of a product. Figure 2.1 shows the relationships among these concepts.

Figure 2.2 presents the feature model for MobileMedia (FIGUEIREDO et al., 2008). It is an SPL for applications that manipulate photo, music, and video on mobile devices. The developers used a previous SPL called MobilePhoto (YOUNG, 2005) as core for developing seven new releases. To implement Mobile Media, the developers extended the core implementation of MobilePhoto by including new mandatory, optional, and alternative features.

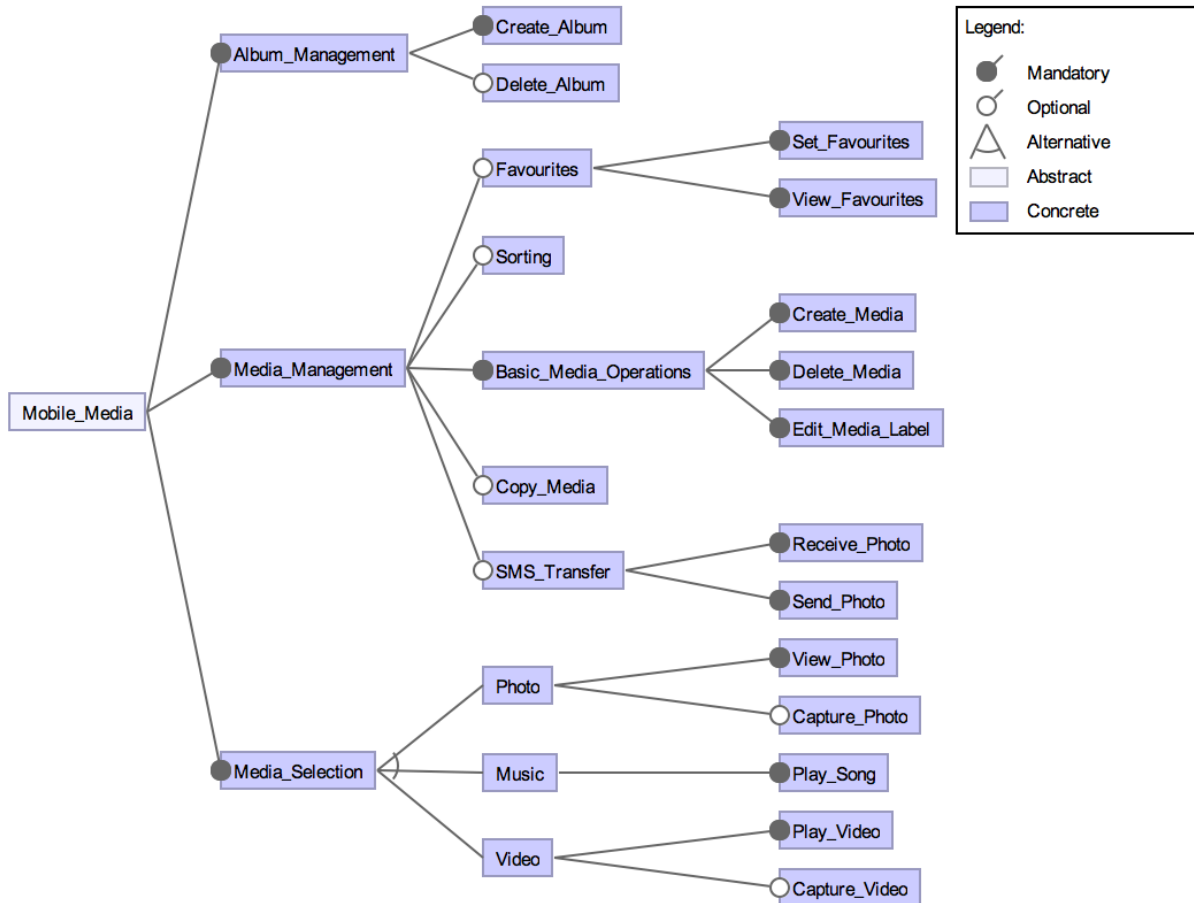


Figure 2.2 Feature Model for MobileMedia.

The mandatory features of the MobileMedia are: create/delete media (photo, music or video), label media, and view/play media. The alternative features are just the types of media supported: photo, music, and/or video. Moreover, the optional features are:

sms transfer, sort media, copy media, set favourite media, delete album, capture photo, and capture video. The mandatory features of MobileMedia are applicable to all the mobile phone devices. The optional and alternative features are configurable on selected mobile phones depending on the API support they provided.

2.1.1 Software Product Line Engineering

The concept of Software Product Line Engineering (SPLE) can be traced back to the late 1960s (APEL et al., 2013). Apel et al. (2013) suggest that “instead of developing software systems from scratch, they should be constructed from reusable parts. Instead of composing software systems always in the same way, they should be tailored to requirements of the customer, where customers can select from a large space of configuration options.”

Krueger (2009) argues that the first generation of SPLs – from the 1980s to 1990s when SPL has gained momentum in the software industry (APEL et al., 2013) – described patterns of software development behavior that later became *Software Product Line Development* (SPLD). Krueger (2009) discussed the new methods behind a new generation of SPL successes. He argued that the early generation made SPL development very different from single system development. However, the new generation case studies reduced these differences considerably.

The main difference between single-system development and SPL development is the change of context: instead of an individual system, stakeholders aim to developing a set of products or portfolio of individual systems that share common and variable components (APEL et al., 2013). This shift suggests a modification of companies strategy: it is necessary to restructure all the sectors, from the infrastructure to the business organization. Moreover, SPL adoption demands management and technical improvements. The integration of these two elements is fundamental for SPL success. Such changes allow the building of a family of products instead of only one application.

SPLE comprises three key activities: Core Asset Development (CAD) aims to develop assets for reuse; Product Development (PD) assembles reusable assets in product instances; and finally, Management handles technical and organizational management (LINDEN; SCHMID; ROMMES, 2007). Pohl, Böckle and Linden (2005) uses the term Domain Engineering (DE) instead of CAD, and Application Engineering (AE) in place of PD.

2.1.1.1 Core Asset Development.

CAD/DE comprises activities that support the development of common assets, and their evolution in response to product feedback, new market needs, and so on (CLEMENTS; NORTHROP, 2001). Figure 2.3 shows the CAD activity along with its outputs and influential contextual factors. CAD is iterative; the rotating arrows suggest that its inputs and outputs affect each other. This context influences the way in which the core assets are produced. For instance, to expand the SPL, the scope may admit new classes of systems to examine possible sources of legacy assets. Restrictions will determine which preexisting assets are candidates for reuse (NORTHROP, 2002).

CAD is divided in five subprocesses: (i) *domain requirements* encompasses all activi-

ties for eliciting and documenting the common and variable requirements of the product line, (ii) *domain design* encompasses all activities for defining the reference architecture of the SPL, (iii) *domain realization* deals with the detailed design and the implementation of reusable software components, (iv) *domain testing* is responsible for the validation and verification of reusable components, and (v) *evolution management* deals with the economic aspects of the SPL, in particular with the market strategy (POHL; BÖCKLE; LINDEN, 2005).

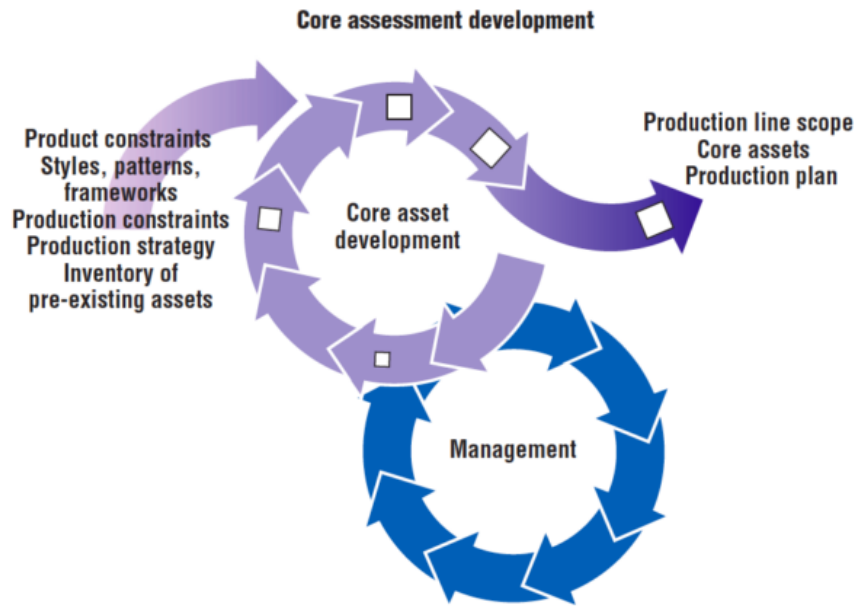


Figure 2.3 Core Asset Development (NORTHROP, 2002)

2.1.1.2 Product Development.

The PD/AE activity has as its main goal the creation of individual products by reusing the core assets previously developed. CAD outputs (product lines scope, core assets and production plan), in conjunction with the requirements for individual products are the main inputs for PD activity. Figure 2.4 illustrates the PD along with its output and influential contextual factors.

PD is divided in four subprocesses: (i) *application requirements* encompasses all activities for developing the application requirements specification, (ii) *application design* encompasses activities for producing the application architecture, (iii) *application realization* creates the considered application, and finally (iv) *application testing* comprises the activities necessary to validate and verify an application against its specification (POHL; BÖCKLE; LINDEN, 2005).

The rotating arrows in Figure 2.4 indicate iteration and involved relationships. This activity has an obligation to give feedback on any problems or deficiencies encountered with the core assets, in order to keep the core asset base in accordance with the products.

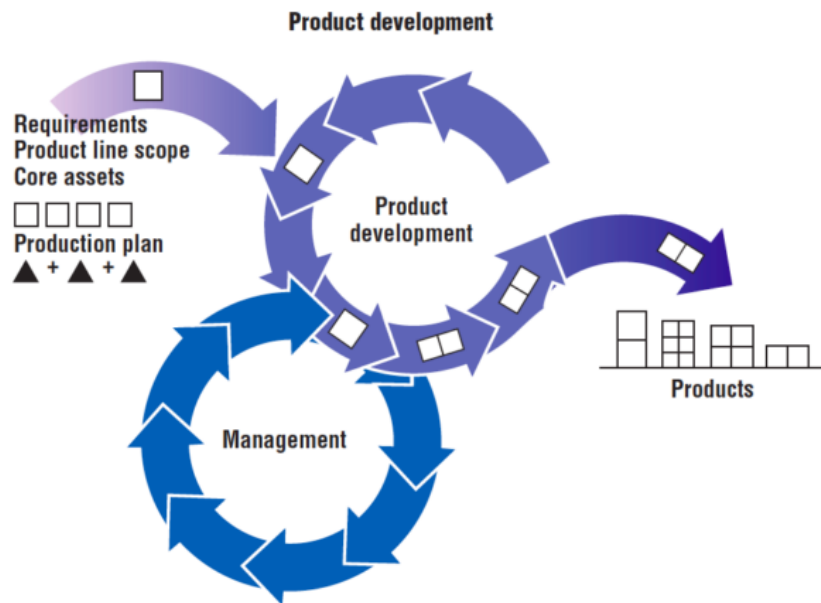


Figure 2.4 Product Development (NORTHROP, 2002)

2.1.1.3 Management.

Management comprises technical management and organizational management. Technical management is responsible for requirements control and the coordination between CAD and PD. Organizational management is responsible for managerial and organizational activities.

The common set of assets and the plan for how they are used to build product do not just materialize without planning, and they certainly do not come free. They require organizational foresight, investment, planning and direction. They require strategic thinking that looks beyond a single product. The disciplined use of the assets to build products does not just happen either. Management must direct, track, and enforce the use of the assets. Software product lines are as much about business practices as they are about technical practices (CLEMENTS; NORTHROP, 2001).

2.1.2 Variability Management

Variability management is a key activity in SPL engineering. Variability provides the required flexibility for product differentiation and diversification (CHEN; BABAR; CAWLEY, 2009). SPL engineering manages dependencies among variants and supports their instantiation in the SPL life cycle (SCHMID; JOHN, 2004). The management includes identifying the commonalities and variations over a set of system artifacts – requirements, architecture, code, and tests (SHULL; BABAR; CHEN, 2010).

Variability management comprises tasks such as elicit and describe variability in software artifacts in the logical and implementation views. The logical representation focuses on the *problem domain* by structuring the features combination. On the other hand, the

variability representation in source code points to the *solution domain* because it implements the features using a specific mechanism (JARING; BOSCH, 2004; SVAHNBERG; GURP; BOSCH, 2005). The increased variability makes the SPL source code more complex than in single systems. As a consequence, the SPL source code is harder to understand (KäSTNER; APEL; KUHLEMANN, 2008).

Moreover, variability management encompasses dependencies establishment and management among different variabilities. It also supports the variabilities exploitation to either build and evolve an SPL (CHEN; BABAR, 2011). In this way, effective methods, techniques, and tools to provide adequate support are necessary because management takes place at different levels of abstraction considering all generic development assets (BOSCH et al., 2002; SINNEMA; DEELSTRA, 2007; CHEN; BABAR, 2011).

2.1.2.1 Variability Modeling.

Variability modeling is one key aspect of the variability management. Several approaches have been introduced for modeling variability in an SPL (KANG et al., 1990; CZARNECKI; EISENECKER, 2000; GOMAA, 2004; SCHMID; JOHN, 2004; POHL; BöCKLE; LINDEN, 2005; DHUNGANA; GRÜNBAACHER; RABISER, 2011).

Feature modeling is a popular technique proposed by Kang et al. (1990) to handle variability modeling. It has been used in different software development paradigms, such as feature-oriented programming (BATORY, 2003; APEL et al., 2013), generative programming (CZARNECKI; EISENECKER, 2000), and model-driven development (TRUJILLO; BATORY; DIAZ, 2007). This technique introduced the concept of feature models.

In the context of SPL, feature models are considered as a standard model to represent variability (KANG et al., 1990; CZARNECKI; EISENECKER, 2000; CHEN; BABAR, 2011). However, feature modeling is just one of many ways used to describe variability. Alternative approaches to model variability are Orthogonal Variability Model (OVM) technique (POHL; BöCKLE; LINDEN, 2005), Decision Models (WEISS; LAI, 1999), and Compositional Variability Management (CVM) (ABELE et al., 2010).

2.1.2.2 Variability Implementation.

Variability implemented in the source code should be in conformance with the variability models early defined in feature modeling and architecture design. To satisfy the SPL final product requirements, variability mechanisms are used in the development process to enable multiple configurations of an SPL.

The literature provides a significant amount of variability implementation mechanisms. Among the existing variability mechanisms, we could enlist conditional compilation, inheritance, parameterization, and overloading as the most widely used ones (SVAHNBERG; GURP; BOSCH, 2005; BOSCH; CAPILLA, 2013). These mechanisms consider the variability in different steps during the development life cycle (SVAHNBERG; GURP; BOSCH, 2005; KIM; HER; CHANG, 2005; MOHAN; RAMESH, 2007; DEELSTRA; SINNEMA; BOSCH, 2009; APEL et al., 2013). Choosing the appropriate variability mechanism to use in the artifacts development is a critical project decision. It is worth to

mention that this investigation concerns the variability issues at the architectural level. Hence, we focus on how the variability implementation affects its representation at this level of SPL projects.

Fritsch et al. (2002) defined that architectural patterns, design patterns, idioms, or guidelines for coding are variability mechanisms as well. There are three main characteristics a mechanism must offer: (i) implementation of the specified options, (ii) a technique to select the options for a certain product configuration, and (iii) the binding time (*Binding Time*). The latter refers to the time a variation was assigned to a variation point and the latest time – during the development – a variation can be found in a variation point (DEELSTRA; SINNEMA; BOSCH, 2009).

Tools such as Feature IDE (MEINICKE et al., 2016) and pure::variants (BEUCHE, 2013) support SPL development and variability management. For instance, the T-Wise Test (HENARD et al., 2014) configuration implemented in FeatureIDE deals with the combinatorial explosion regarding the products configurations.

2.2 SOFTWARE ARCHITECTURE

The software engineering community provided several definitions for software architecture (SA), each one emphasizing specific aspects of it (SHAW; GARLAN, 1996; GARLAN, 2000; BASS; CLEMENTS; KAZMAN, 2003; CLEMENTS et al., 2010; TAYLOR; MEDVIDOVIC; DASHOFY, 2009).

Shaw and Garlan (1996) stated that “the architecture of a system defines that system in terms of computational components and interactions among those components”. Common terminology for architectural design elements (*Design Element*) included “component” as a locus of computation, “connector” as a locus of interaction between components, and “configuration” as a set of attached components and connectors that provides some functionality. Accordingly, Bass, Clements and Kazman (2003) defined SA as the system structure, which consists of software elements, externally visible properties, and the relationships among elements (BASS; CLEMENTS; KAZMAN, 2003).

Definitions have evolved to include the importance of design decisions and rationale besides the focus on the top-level decomposition of a system (or a set of systems) into its main components. Taylor, Medvidovic and Dashofy (2009) emphasize the role of principal design decisions and define SA as the set of principal design decisions made during the development and evolution. For this reason, making the SA sound and persistent is critical for using the potential it offers as an enabler for software development in an efficient and effective way, particularly in scenarios of increasing system size, evolution, and complexity (ROST et al., 2013).

The ISO/IEC/IEEE 42010 standard for architecture description (ISO/IEC/IEEE. . . , 2011) defines SA as “the fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution”.

A compilation of definitions for SA can be found in SEI website¹. In this work, we use the standard definition (ISO/IEC/IEEE. . . , 2011), complemented by the one from (TAYLOR; MEDVIDOVIC; DASHOFY, 2009).

¹<<https://bit.ly/2S3hTSt>>

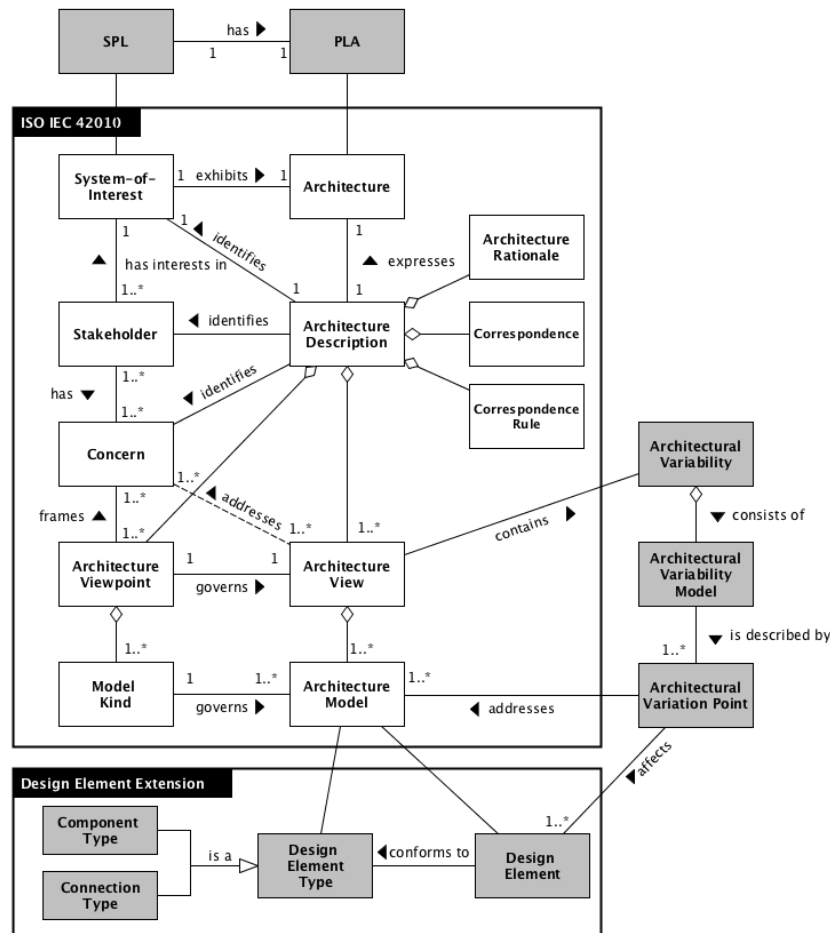


Figure 2.5 Excerpt from ISO/IEC/IEEE 42010 and design elements – adapted from (THIEL; HEIN, 2002b)

SA is important for different stakeholders (*Stakeholder*) with different perspectives or viewpoints. When explicitly documented, SA is relevant for several activities such as software development, analysis, maintainability, and evolution. Stakeholders of an SPL are parties with interests in that SPL. They have concerns (*Concern*) with respect to the SPL-of-interest considered in relation to its domain. A concern could be held by one or more stakeholders. They arise throughout the life cycle from SPL needs and requirements, from design choices and from implementation considerations.

2.2.1 Architecture Descriptions

The Joint Technical Committee ISO/IEC in cooperation with the Software and Systems Engineering Standards Committee of the Computer Society of the IEEE prepared the ISO/IEC/IEEE 42010² (ISO/IEC/IEEE..., 2011) to address the creation, analysis, and sustainment of architectures of systems by means of architecture descriptions.

²<<https://www.iso.org/standard/50508.html>>

An *Architecture Description* is the set of artifacts that document an architecture. It comprises one or more architecture views. An *Architectural View* is a representation of the software architecture that addresses one or more stakeholders' concerns as can be seen in Figure 2.5.

Moreover, an architecture view describes the architecture of the SPL-of-interest according to an architecture viewpoint. A *Viewpoint* defines the perspective from which an architecture view is taken. It defines: how to construct and use an architecture view; the information that should appear in the architecture view; the modeling techniques for expressing and analyzing the information; and a rationale for these choices. An architecture viewpoint frames one or more concerns. A concern can be framed by more than one viewpoint.

For instance, Clements et al. (2010) in their approach to SA description (mostly compliant with the ISO/IEC/IEEE 42010 standard) present three kinds of architectural viewpoint (or viewtype, as they define it): Module, Component-and-Connector and Allocation. The Module views are concerned with documenting a system's principal units of implementation. Component-and-connector views document the system's unity of execution. Allocation views document the relations between a system's software and nonsoftware resources of the development and execution environments.

An architectural view consists of one or more *Architecture Models* that uses appropriate modelling conventions notation to address the concerns. These conventions are specified by the *model kind* governing that model. Within an architecture description, an architecture model can be part of more than one architecture view. In the following subsections, we will present architecture view models used in this thesis.

2.2.1.1 Development Views.

Kruchten (1995) proposed the 4+1 architectural model to represent SA from different viewpoints. The model is organized in five views, as depicted in Figure 2.6: logical, development, process, physical, and the fifth view combines the former ones as a means to illustrate and explain the overall architecture.

The development view, at the top right corner of Figure 2.6 presents the development view that illustrates a system from a developer's perspective. This view is also known as the implementation view. It describes the system components. Components are represented as collections of source code artifacts (*e.g.* classes and packages) and system connectors as relations between these system components (*e.g.* calls, uses, sets) (KRUCHTEN, 1995).

Listing 2.1 describes an excerpt of the MobileMedia source code. The methods `getBytesFromImage()` and `addImageData()` were implemented using `#ifdef` annotation. When a developer select the `#ifdef includeSmsFeature` these methods in the configuration they will be present on the product source code.

```
// #ifdef includeSmsFeature
/* [NC] Added in scenario 06 */
public byte[] getBytesFromImage(Image img){
    int w = img.getWidth();
    int h = img.getHeight();
```

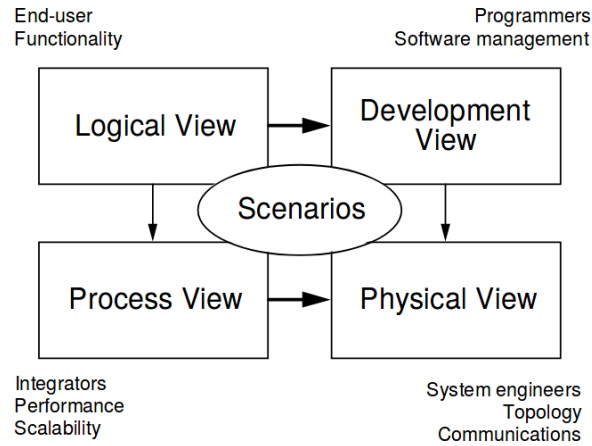


Figure 2.6 The 4+1 view model, extracted from (KRUCHTEN, 1995)

```

int data_int[] = new int[ w * h ];
img.getRGB( data_int, 0, w, 0, 0, w, h );
byte[] data_byte = new byte[ w * h * 3 ];
for ( int i = 0; i < w * h; ++i ){
    int color = data_int[ i ];
    int offset = i * 3;
    data_byte[ offset ] = ( byte ) ( ( color & 0xff0000 ) >> 16 );
    data_byte[ offset + 1 ] = ( byte ) ( ( color & 0xff00 ) >> 8 );
    data_byte[ offset + 2 ] = ( byte ) ( ( color & 0xff ) );
}
return data_byte;
}

public void addImageData(String photoname, Image imgdata, String albumname)
    throws InvalidImageDataException, PersistenceMechanismException {
    try {
        byte[] data1 = getByteFromImage(imgdata);
        addMediaArrayOfBytes(photoname, albumname, data1);
    } catch (RecordStoreException e) {
        throw new PersistenceMechanismException();
    }
}
}
//endif

```

Listing 2.1 Excerpt of MobileMedia source code

DSM. The Design Structure Matrix (DSM; also referred to as dependency structure matrix) is a visual representation of a system in the form of a square matrix representing relations between the system elements (EPPINGER; BROWNING, 2012). The system elements are labeled in the rows to the left of the matrix and in the columns above the matrix. These elements can represent packages, classes, modules and product components.

Figure 2.7 presents the MobileMedia’s DSM. It describes the packages relationships. For instance, the package `datamodel` calls 18 methods in the package `controller`.

Module Dependency Graph. Module Dependency Graph (MDG) represents the software elements (such as packages, classes, modules, and so on) and relationships used

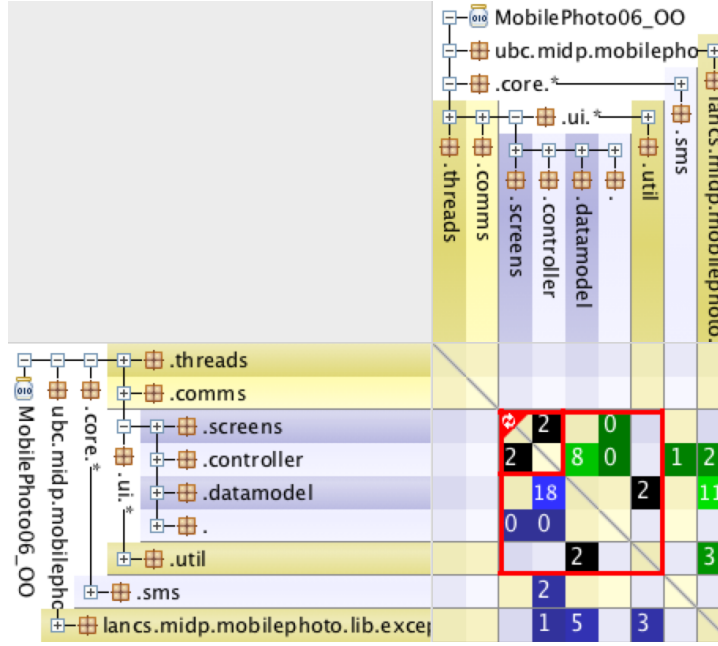


Figure 2.7 DSM for MobileMedia

in the architectural description of traditional systems (MANCORIDIS et al., 1999). Figure 2.8 presents MobileMedia packages and its relationship.

2.2.2 Architecture Variability

The Software Architecture discipline treats the variability as a quality attribute and a cross-cutting concern (GALSTER; AVGERIOU, 2011b; GALSTER et al., 2013). Moreover, SA considers variability in a broader scope and acknowledges that variability is a *Concern* for different stakeholders, and in turn affects other concerns. Figure 2.9 presents the architecture variability extension.

Variability is a relevant characteristic of the architectures of software systems (HILLIARD, 2010; GALSTER; AVGERIOU, 2011b) – either single systems, SPL, and system of systems. It is a key fact of “most, if not all systems” and therefore a relevant concern for the architectures of those systems (HILLIARD, 2010) – that should be early managed and identified during architecting over discovering and addressing it later in the life cycle (THIEL; HEIN, 2002a).

As defined by the ISO/IEC/IEEE 42010, the description of an architecture should be organized into multiple views. Each view addresses one or more stakeholder concerns. In the context of SPL, each architecture view contains *Architectural Variability* to cope with the variety of characteristics that must be implemented for the different SPL members. Architectural variability reflects the existence of alternative design options that could not be bound during architectural modeling. Architectural variability is usually expressed by a set of architectural variation points (see Figure 2.9).

Architectural variability can be summarized in an *Architectural Variability Model*, de-

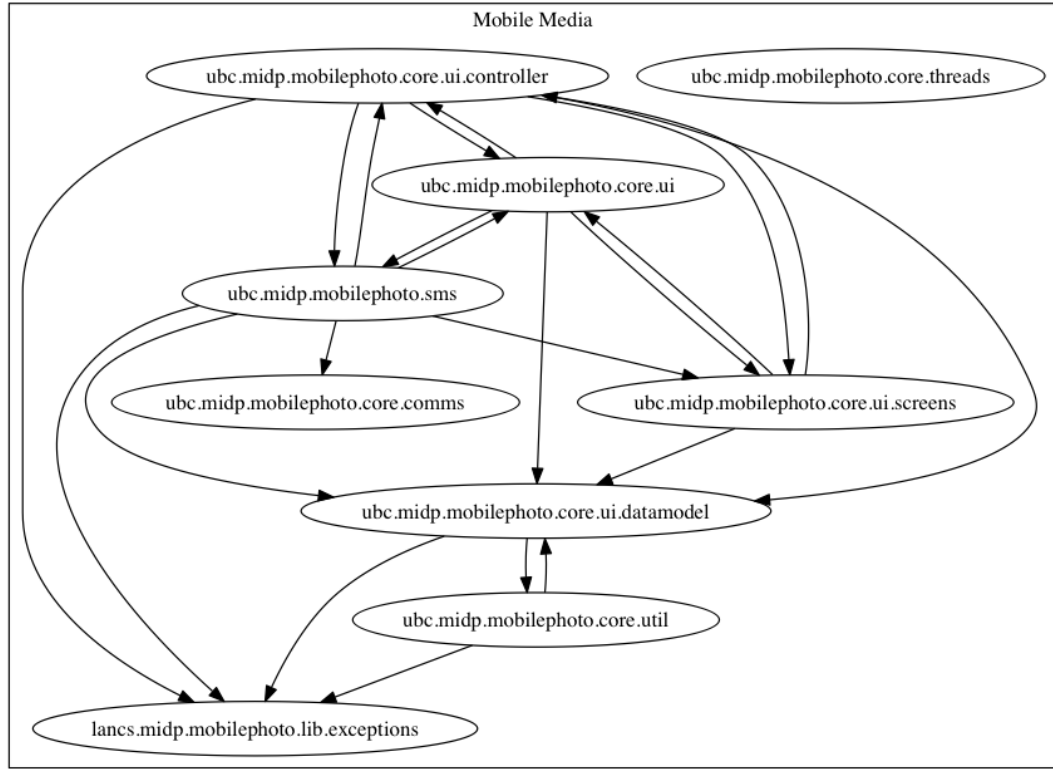


Figure 2.8 MDG for MobileMedia

scribed by a set of the *Architectural Variation Point*. An architectural variation point represents the architectural elements responsible for describing the PLA variability. It shows part of the architectural solution of the architectural variability. The set of architectural variation points needs to be consistently resolved and bound to concrete design options.

Figure 2.10 presents the architectural variability model for MobileMedia proposed by Shatnawi, Seriai and Sahraoui (2016). The large boxes denote design decisions (dependencies). For instance, core architecture refers to components that should be selected to create any concrete PLA. In MobileMedia, there is one mandatory component manipulating the base controller of the product. This component has two variants. A group of Multi Media Stream, Video Screen Controller, and Multi Screen Music components represents design decision regarding the implementation of a alternative feature.

Variability is represented and facilitated through the SA. In this way, variability in the architecture is a complex concept and dealing with it is a multi-faceted activity that should be treated as first-class citizen in software architecture (GALSTER et al., 2013). Consequently, to develop the appropriate support for architects to deal with variability, it is worth to comprehend the various problems software architects come across when attempting to carry out variability-related tasks (GALSTER; AVGERIOU, 2011a).

So far, variability has primarily been addressed in the SPL domain (GALSTER; AVGERIOU, 2011b, 2011a; CHEN; BABAR, 2011; GALSTER et al., 2013). In this

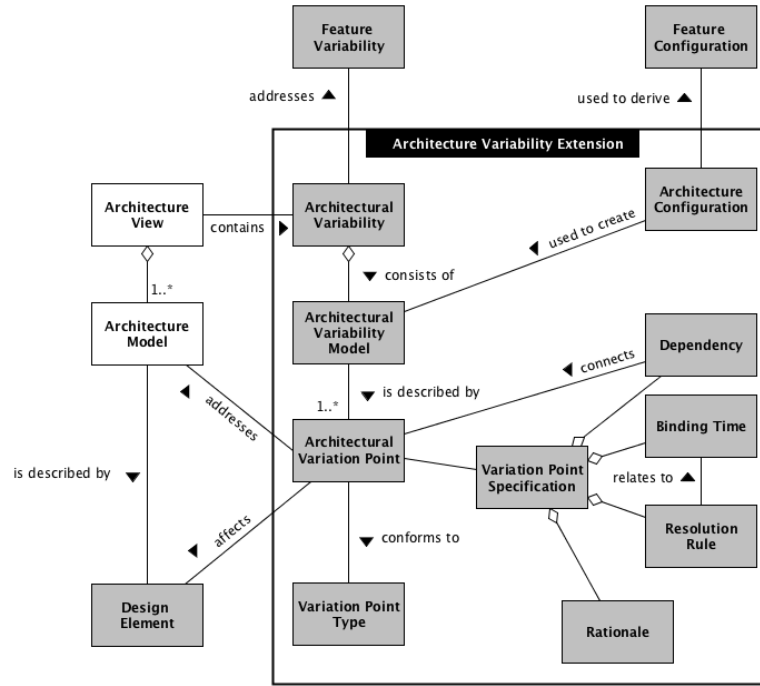


Figure 2.9 Excerpt from Feature Variability Extension – adapted from (THIEL; HEIN, 2002b)

way, to address the variability in the architectural level of SPL projects, the SPL community introduced the notion of Product Line Architecture (AHMED; CAPRETZ, 2008). PLA captures the central design of all products including variability and commonalities of several products instances (VERLAGE; KIESGEN, 2005).

Haider, Woods and Bashroush (2018) performed a SLR to capture and summarize the state-of-the-art in representing variability in SA. The authors states that variability representation at the SA is mostly related to SPL context. Moreover, they found that UML (including various extensions) and Architecture Description Languages (ADL) were the most used notations to represent variability in SA.

2.3 PRODUCT LINE ARCHITECTURE

A *Product Line Architecture* – or Software Product Line Architecture (SPLA) – is the core architecture that represents a high-level design for all the products of an SPL, including variation points and variants documented in the variability model (POHL; BÖCKLE; LINDEN, 2005). Because the development of an SPL involves the implementation of different structures, processes, interfaces and activities, it is relevant for SPL practitioners to pay sufficient attention to its architecture.

A PLA stands for a SPL as much as a single system software architecture stands for a SPL product instance. The PLA provides for an explicit high-level representation that is variability-aware. This means that besides an overall description of the “core” elements and relationships present in the architecture of all SPL products, the PLA must include

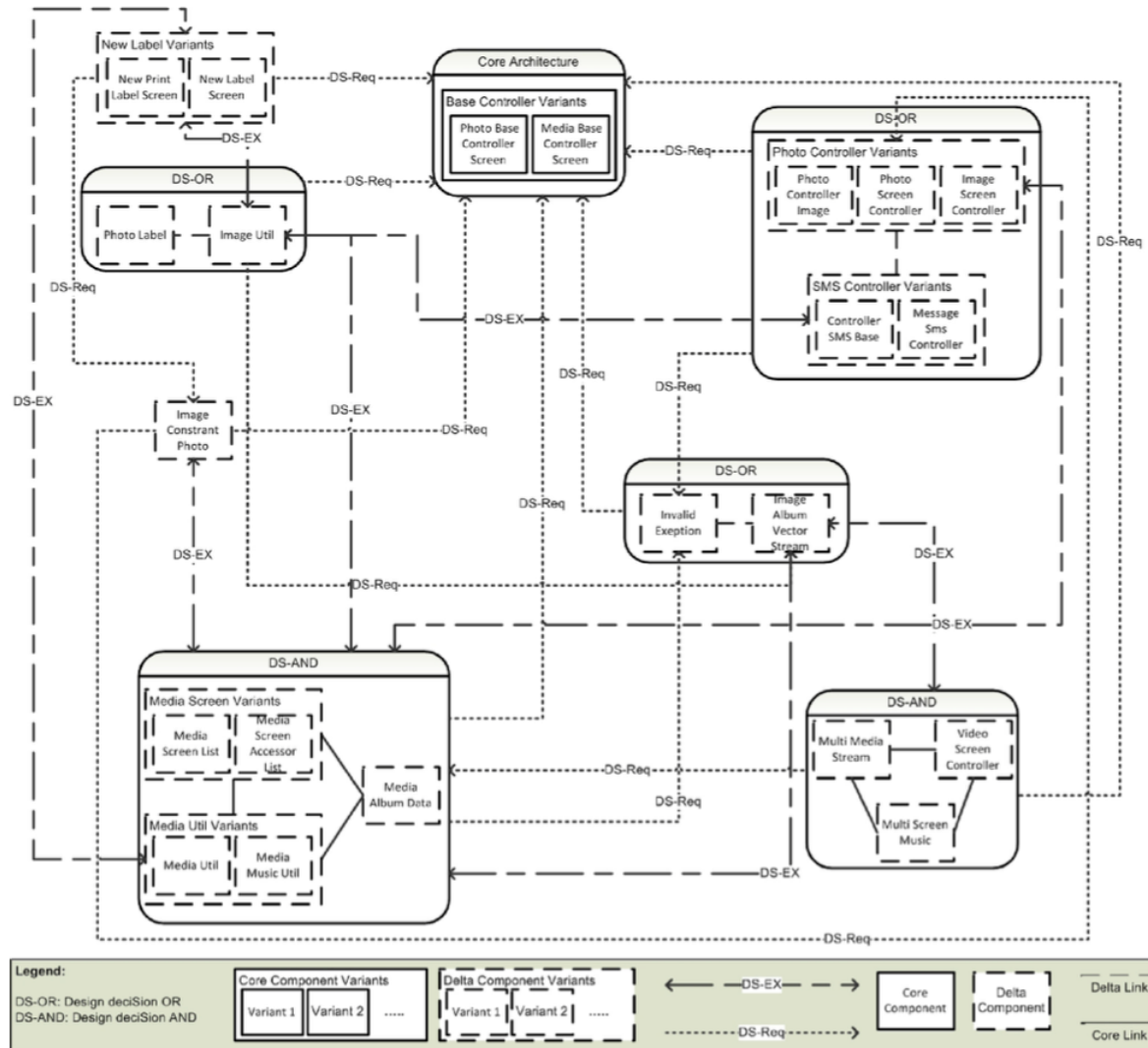


Figure 2.10 Variability Model for MobileMedia (SHATNAWI; SERIAI; SAHRAOUI, 2016)

a high-level description of variable elements and their relationships that will be specific to some products after product configuration. Different product instances share the SPL “core architecture”.

According to Martínez-Fernández et al. (2013), the terms Reference Architecture (RA) and PLA are sometimes used indistinctly. The term RA is used to refer to “a core architecture that captures the high-level design for the application of the SPL” (POHL; BÖCKLE; LINDEN, 2005) or “just one asset, albeit an important one, in the SPL asset base” (CLEMENTS; NORTHROP, 2001). However, out of the SPL context, RA and PLA are considered different types of artifacts (NAKAGAWA; ANTONINO; BECKER, 2011; ANGELOV; GREFEN; GREEFHORST, 2012; EKLUND et al., 2012; GALSTER et al., 2013). Angelov, Grefen and Greefhorst (2012) claim that a PLA is a RA whereas not every RA is a PLA. A PLA is just one SPL asset (CLEMENTS; NORTHROP, 2001). A

RA provides standardized solutions for a broader domain whereas a PLA provides a standardized solution for a smaller subset of the software system of a domain (NAKAGAWA; ANTONINO; BECKER, 2011). Moreover, PLAs address variability points and more formal specification to ensure clear and precise behavior specifications at well-specified extension points (ANGELOV; GREFEN; GREEFHORST, 2012). In contrast, RAs have less focus on capturing variation points (NAKAGAWA; ANTONINO; BECKER, 2011; ANGELOV; GREFEN; GREEFHORST, 2012; EKLUND et al., 2012).

A *PLA Description* is an architecture description enriched with explicit information about the variability model of the SPL. This means that PLA architecture models must be concerned with the representation of *Architectural Variation Points*.

For PLAs, architectural views are extended with architectural variability representation. In this work, variability information is represented in development views and architectural elements such as packages and classes.

2.4 SOFTWARE ARCHITECTURE RECOVERY

The concern about recovering architectural information (or “high level” design information) started in the 1990s along with the formal study of software architecture. Early research work investigated and integrated reverse engineering technology with architectural representations (CHIKOFSKY; CROSS, 1990; WATERS; CHIKOFSKY, 1994; HARRIS; REUBENSTEIN; YEH, 1995).

Software Architecture Recovery (SAR) (GALL et al., 1996; MENDONCA; KRAMER, 1996) stands for the research field concerned with the use and integration of concepts from reverse engineering and software architecture.

According to Gall et al. (1996), recovering the architecture of software systems requires more than just reverse engineering tools. It is necessary to balance information about design decisions and logical functions with informal domain knowledge, domain standards, and developers coding guidelines.

The majority of the studies identified by Ducasse and Pollet (2009), considered SAR as the process of extracting architectural information from software systems. The most used techniques that support architecture recovery extracts information from the system source code (GARCIA; IVKOVIC; MEDVIDOVIC, 2013).

According to Mendonca and Kramer (1998), SAR can facilitate system understanding and maintenance. The authors proposed a concern regarding the use of SAR to support product families focusing on the recovery of legacy systems from the same domain. They argued that SAR helps in the identification of commonalities within a family of related system and also fosters the possibility of system structure reuse in the development of new systems.

Some research uses the term “software architecture reconstruction” as a synonym for SAR, for instance, early work of Kazman and Carrière (1999). Deursen et al. (2004) state that SA reconstruction is the process of obtaining a documented architecture for an existing system, although, such a reconstruction can make use of any possible resource. Ducasse and Pollet (2009) defined SA reconstruction as a reverse engineering approach that aims at reconstructing architectural views of a software application. The

authors also identified other terms in the literature to refer to reconstruction: reverse architecture/architecting, architecture extraction/mining/recovery/discovery.

2.4.1 Processes

There are three processes for architecture reconstruction (DUCASSE; POLLET, 2009): bottom-up, top-down and hybrid processes. Often, “recovery” refers to a bottom-up process, while “discovery” refers to a top-down process.

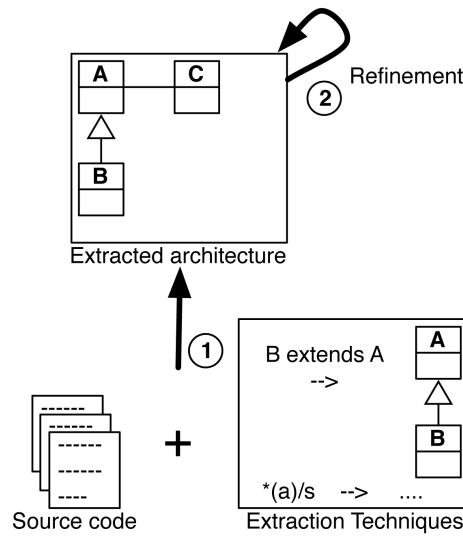


Figure 2.11 A bottom-up process: From source code, (1) views are extracted and (2) refined (DUCASSE; POLLET, 2009).

Bottom-up process. Bottom-up processes **recover** the architecture using low-level knowledge (*e.g.* source code structural information) (DUCASSE; POLLET, 2009). They raise the abstraction level in a progressive way, starting with source code – through a series of chunking and concept assignments steps (TILLEY; PAUL; SMITH, 1996) – until reaching a high-level understanding of the application.

Moreover, bottom-up processes (see Figure 2.11) – also called architecture recovery processes – are related to the description of extract-abstract-present cycle (TILLEY; PAUL; SMITH, 1996). With the support of source code analysis, the authors suggest a repository population by querying to yield abstract system representations in an interactive form to reverse engineers.

In the context of this thesis, we focus on using and adapting existing bottom-up processes, techniques, and tools to recover PLAs. We create a source code model based on low-level knowledge (source-code). We raise the abstraction level and we represent the PLA using development views.

Top-Down Process. Top-down processes (see Figure 2.12) aim to **discover** the architecture by formulating conceptual hypotheses. In other words, top-down pro-

cesses are opposite from bottom-up processes. They start with high-level knowledge (*e.g.* requirements of architectural styles) until matching them to the source code (DUCASSE; POLLET, 2009).

Moreover, the top-down approach starts with a pre-existing notion of the system functionality and progresses to the system components. The refinement method includes hypotheses creation, verification, and modification until it is possible to explain the entire system (TILLEY; PAUL; SMITH, 1996).

The extraction of logical views depends on the analysis of requirements models, use cases, and activity models. In such cases, top-down processes are used because they start with high-level knowledge and aim to discover the architecture by formulating hypotheses (DUCASSE; POLLET, 2009).

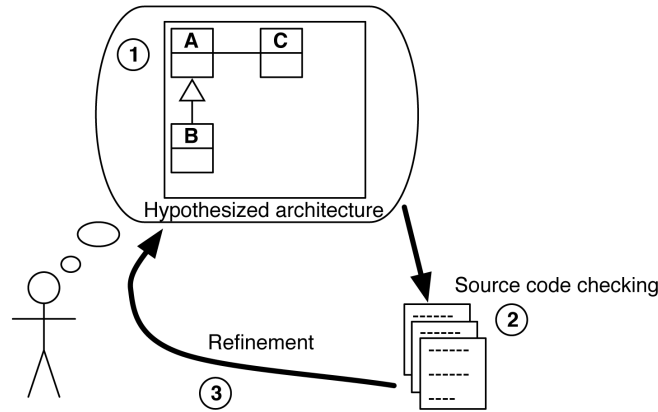


Figure 2.12 A top-down process: (1) A hypothesized architecture is defined. (2) The architecture is checked against source code. (3) The architecture is refined (DUCASSE; POLLET, 2009).

Hybrid Process.

The combination of bottom-up and top-down processes allow the implementation of hybrid processes. In this way, techniques can be used to abstract the low-level knowledge – **recovered** from bottom-up processes. The refinement of high-level knowledge – **discovered** from top-down processes – can be confronted with the previously extracted views (DUCASSE; POLLET, 2009).

Hybrid processes compare the conceptual or prescriptive architecture against the concrete or descriptive architecture (TAYLOR; MEDVIDOVIC; DASHOFY, 2009). Moreover, hybrid approaches often explore architectural hypothesis provided by top-down techniques combined with bottom-up reverse engineering strategies (PASHOV; RIEBISCH, 2004).

2.4.2 Techniques

The recovery of different software architecture views requires the use of different extraction techniques (STAVROPOULOU; GRIGORIOU; KONTOGIANNIS, 2017). Most

SAR approaches adopt static analysis techniques to recover design-time information in module views (CLEMENTS et al., 2010) or development views (KRUCHTEN, 1995). With respect to behavior and runtime information, dynamic analysis techniques can be used to reconstruct component-and-connector views, *e.g.* process views.

For recovering development views, most SAR techniques are based on source code static analysis and clustering. They group entities into clusters, where each cluster represents a component of the system’s architecture (STAVROPOULOU; GRIGORIOU; KONTOGIANNIS, 2017).

Ducasse and Pollet (2009) classify techniques with respect to their automation level:

Quasi-manual techniques.

Quasi-manual techniques provide support on the understanding manually extracted architectural elements. In this context, researchers proposed slightly assisted quasi-manual techniques such as Constructed-Based Techniques and Exploration-based techniques. The former reconstructs the software architecture by manually abstracting low-level knowledge. The latter gives reverse engineers an architectural view of the system by guiding them through the highest level artifacts of the implementation. In such cases, the architectural view is closely related to the developer’s view (DUCASSE; POLLET, 2009).

Semiautomatic techniques.

Semiautomatic techniques automate repetitive aspects of SAR. Some examples of such techniques are (i) Abstraction-based techniques that aim to map low-level concepts with high-level concepts and (ii) Investigation-based techniques that map high-level concepts with low-level concepts. The high-level concepts considered cover a wide area from architectural descriptions and styles to design patterns and features (DUCASSE; POLLET, 2009).

Quasi-automatic techniques.

Quasi-automatic techniques automate the SAR with tool support. Some examples are (i) Formal concept analysis that is a branch of lattice theory, (ii) Clustering algorithms which identify groups of objects whose members are similar in some way, (iii) Dominance analysis for identifying related parts in an application, and (iv) Dependency Structure Matrix which is used to analyze architectural dependencies in software by showing compact manner dependencies between source code entities such as classes and packages (DUCASSE; POLLET, 2009).

2.4.3 Tools

Tools can be used to support the SAR processes and techniques. They allow the SAR automation and eliminate repetitive tasks. Some tools focus on software structure extraction such as Stan4J, Analizo, and Struct101. Other tools focus on providing visualization of the recovered information. For instance, tools such as Archvis, Rigi, Graphviz and CodeCrawler are used to visualize graph representations of software views.

Some tools such as Bunch, ACDC, and Arcade implements clustering algorithms for presenting and grouping source code entities as boxes. Moreover, tools like SoftArch

supports both static and dynamic visualization of software architecture components at varying levels of abstraction.

2.4.4 Product Line Architecture Recovery

As stated by Medvidovic, Egyed and Grünbacher (2003), the term recovery refers to a bottom-up process. Therefore, architecture variability recovery comprises tasks to recovering variability-aware architectural models, often from low-level artifacts such as the source code of software products. In the context of this thesis, we are interested in architecture variability recovery for SPL, or PLA recovery. This means that architectural abstractions must be variability-aware and aligned with the SPL terminology. In Chapter 3, we exploit and discuss PLA recovery in detail.

2.5 REUSE ASSESSMENT

PLAs can be used to assess some quality dimensions of the SPL such as reusability. For instance, reuse assessment comprises on reflecting the presence of design component characteristics that allow a design to be reapplied to a new component without effort and preventing the propagation of errors to other variants. In the context of this work, PLA components and their relationships are used to support SPL reuse assessment.

According to Cordeiro and OliveiraJr (2018), software metrics can support PLA evaluations. The use of software metrics helps in the understanding, controlling and improvement of activities and/or artifacts (FENTON; BIEMAN, 2014). To select the right metrics to our thesis, we searched the literature to identify metrics related to PLA and SAR. Table 2.1 describes the metrics used in this thesis to support reuse assessment of SPLs. Some metrics have been defined for the quality assessment of PLAs (ZHANG et al., 2008). Their definitions are provided below.

SSC (Structure Similarity Coefficient) is used to measure the similarity between PLA components, while **SVC** (Structure Variability Coefficient) 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.

RSC (Relation Similarity Coefficient) is used to measure the similarity between PLA relations, and **RVC** (Relation Variability Coefficient) 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.

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.

The **CRR** (Component Reuse Rate) is used to calculate the reuse rate of each PLA element. The $Ex(M_i)$ - returns 1, if component i is present in the product architecture, 0 otherwise. On the other hand, the **RRR** (Relationship Reuse Rate) is used to calculate the reuse rate of each PLA relationship. The $Ex(M_j)$ - returns 1, if relationship j is

present in the product architecture, 0 otherwise.

The metrics **CM** Class Mandatory calculates the number of classes implementing the mandatory features and **CO** Class Optional calculates the number of classes implementing the optional features.

Besides, the metrics **OP** Optional Relation and **MR** Mandatory Relation use the same principle to calculate the number of mandatory and optional relations.

The **PLTV** (PLTotalVariability) is a metric that estimates the PLA variability based on the metrics **CO** and **OR** results.

Table 2.1 Metrics used to evaluate the PLA.

Metric	Description	Formula	Source
SSC	SSC calculates the overall similarity between PLA components.	$\frac{ C_C }{ C_C + C_V }$	(ZHANG et al., 2008)
RSC	RSC calculates the overall similarity between PLA relationships.	$\frac{ R_C }{ R_C + R_V }$	(ZHANG et al., 2008)
SVC	SVC calculates the overall variability between PLA components.	$\frac{ C_V }{ C_C + C_V }$	(ZHANG et al., 2008)
RVC	RVC calculates the overall variability between PLA relationships.	$\frac{ R_V }{ R_C + R_V }$	(ZHANG et al., 2008)
CRR	Calculates the component reuse rate of each component of the PLA	$\frac{\sum_i Ex(M_i)}{ M } \times 100\%$	(ZHANG et al., 2008)
RRR	Calculates the relationship reuse rate of each component of the PLA	$\frac{\sum_j Ex(M_j)}{ M } \times 100\%$	(ZHANG et al., 2008)
CM	Calculates the number of packages or classes implementing the mandatory features	$\sum C_C$	(OLIVEIRA et al., 2008)
CO	Calculates the number of packages or classes implementing the optional features	$\sum C_V$	(OLIVEIRA et al., 2008)
MR	Calculates the number of relationships implementing the mandatory features	$\sum R_C$	(OLIVEIRA et al., 2008)
OR	Calculates the number of relationships implementing the optional features	$\sum R_V$	(OLIVEIRA et al., 2008)
PLTV	Estimates the number of variable components found on PLA	$\sum R_V + \sum C_V$	(OLIVEIRA et al., 2008)

Legend: C_C - Total number of Common Components; R_C - Total number of Common Relationships; C_V - Total number of Variable Components; R_V - Total number of Variable Relationships; M - Total number of variants; $Ex(M_i)$ - returns 1, if component i is present in the product architecture, 0 otherwise. $Ex(M_j)$ - returns 1, if relationship j is present in the product architecture, 0 otherwise.

Table 2.2 shows the metrics for MobileMedia. The project presents high number of variable elements (**CO** - 52) and relations (**OR** - 148) against a low number of mandatory elements (**CM** - 7) and relations (**MR** - 3). As a consequence, we highlight the high structure variability coefficient (**SVC** - 0.88) and relation variability coefficient (**SVC** - 0.98).

Table 2.3 presents the component reuse rate (**CRR**) values for the packages of MobileMedia. For instance, the package `lancs.midp.mobilephoto.lib.exceptions` presents

Table 2.2 Recovered Metrics for MobileMedia

SSC	SVC	RSC	RVC	CO	OR	CM	MR
0.12	0.88	0.02	0.98	52	148	7	3

CRR of 87.5% while the packages `ubc.midp.mobilephoto.core.ui` presents CRR of 100% indicating that the package is implemented in all the variants.

Table 2.3 CRR for MobileMedia

Package	CRR
<code>lancs.midp.mobilephoto.lib.exceptions</code>	87.5
<code>ubc.midp.mobilephoto.core.comms</code>	37.5
<code>ubc.midp.mobilephoto.core.threads</code>	100.0
<code>ubc.midp.mobilephoto.core.ui</code>	100.0
<code>ubc.midp.mobilephoto.core.ui.controller</code>	100.0
<code>ubc.midp.mobilephoto.core.ui.datamodel</code>	100.0
<code>ubc.midp.mobilephoto.core.ui.screens</code>	100.0
<code>ubc.midp.mobilephoto.core.util</code>	100.0
<code>ubc.midp.mobilephoto.sms</code>	37.5

Table 2.4 presents the relationships reuse rate (RRR) values for the MobileMedia project. For instance, the relationship between Package A (`ubc.midp.mobilephoto.core.ui`) and Package B (`ubc.midp.mobilephoto.core.ui.controller`) presents RRR of 100%, which means that this relationship is implemented in all the variants.

2.6 CHAPTER SUMMARY

This chapter introduced general concepts of SA (and its recovery) and SPL (including detailed discussion about PLA). In a nutshell, SPL exploits commonalities among products to reduce costs, time to market, and improve the software quality. On the other hand, SPL provides the variability management allowing the organization to achieve economies of scope and provides the capability of mass customization.

Because the development of an SPL involves the implementation of different structures, processes, interfaces and activities, it is relevant for SPL practitioners to pay sufficient attention to its architecture. In this context, the PLA enables the maximization of the architecture reuse across several products. PLA recovery provides information to support the understanding of the variability mechanism implemented. Next Chapter, we discuss the current state-of-the-art on the relationship between SAR and PLA, and leverage existing gaps towards PLA recovery.

Table 2.4 RRR for MobileMedia

Package A	Package B	RRR
ubc.midp.mobilephoto.core.ui	ubc.midp.mobilephoto.core.ui.controller	100.0
ubc.midp.mobilephoto.core.ui	ubc.midp.mobilephoto.core.ui.datamodel	100.0
ubc.midp.mobilephoto.core.ui	ubc.midp.mobilephoto.core.ui.screens	37.5
ubc.midp.mobilephoto.core.ui	ubc.midp.mobilephoto.sms	25.0
ubc.midp.mobilephoto.core.ui.controller	ubc.midp.mobilephoto.core.ui	100.0
ubc.midp.mobilephoto.core.ui.controller	ubc.midp.mobilephoto.core.ui.datamodel	100.0
ubc.midp.mobilephoto.core.ui.controller	ubc.midp.mobilephoto.core.ui.screens	100.0
ubc.midp.mobilephoto.core.ui.controller	lancs.midp.mobilephoto.lib.exceptions	87.5
ubc.midp.mobilephoto.core.ui.controller	ubc.midp.mobilephoto.sms	37.5
ubc.midp.mobilephoto.core.ui.datamodel	ubc.midp.mobilephoto.core.util	100.0
ubc.midp.mobilephoto.core.ui.datamodel	lancs.midp.mobilephoto.lib.exceptions	87.5
ubc.midp.mobilephoto.core.ui.screens	ubc.midp.mobilephoto.core.ui.datamodel	100.0
ubc.midp.mobilephoto.core.ui.screens	lancs.midp.mobilephoto.lib.exceptions	62.5
ubc.midp.mobilephoto.core.ui.screens	ubc.midp.mobilephoto.core.ui	25.0
ubc.midp.mobilephoto.core.ui.screens	ubc.midp.mobilephoto.core.ui.controller	25.0
ubc.midp.mobilephoto.core.util	ubc.midp.mobilephoto.core.ui.datamodel	100.0
ubc.midp.mobilephoto.core.util	lancs.midp.mobilephoto.lib.exceptions	87.5
ubc.midp.mobilephoto.sms	lancs.midp.mobilephoto.lib.exceptions	37.5
ubc.midp.mobilephoto.sms	ubc.midp.mobilephoto.core.comms	37.5
ubc.midp.mobilephoto.sms	ubc.midp.mobilephoto.core.ui	37.5
ubc.midp.mobilephoto.sms	ubc.midp.mobilephoto.core.ui.controller	37.5
ubc.midp.mobilephoto.sms	ubc.midp.mobilephoto.core.ui.datamodel	37.5
ubc.midp.mobilephoto.sms	ubc.midp.mobilephoto.core.ui.screens	37.5

To dream is to wake up on the inside. – Mário Quintana

A SYSTEMATIC MAPPING STUDY ON PRODUCT LINE ARCHITECTURE RECOVERY

Product Line Architecture recovery brings an important additional benefit to those commonly credited to the architecture recovery of single systems (DUCASSE; POLLET, 2009): the possibility of managing variability at the architectural level (SHATNAWI; SERIAI; SAHRAOUI, 2015). It supports keeping SPL assets such as architectural documentation and design artifacts up-to-date (PINZGER et al., 2003).

PLA recovery must support the identification of commonalities and variability within the SPL products from one or more sources of information. For instance, PLA recovery processes may focus on representative members of the product line resulting in the architectural description for each product line member (EIXELBERGER, 2000).

However, no literature reviews which investigated SAR with focus on SPL, PLA or variability management were found. In this context, we performed a SMS to gather data and evidence about research work that combines PLA and SAR.

This Chapter presents an overview to summarize and categorize the state-of-the art on PLA Recovery. Section 3.2 details the review process. Section 3.3 presents the outcomes and findings. Section 3.4 presents discussion the findings and describes the threats to validity of this work. We discuss the related work in Section 3.5. Section 3.6 presents the chapter summary.

3.1 MOTIVATION

According to Shatnawi, Seriai and Sahraoui (2016), few approaches reported in the literature support PLA recovery. The recovery of variability is one of their limitations. Moreover, they recovered only some variability aspects and missed the recovery of the whole PLA.

We define the goal of our study using the Goal-Question-Metric (GQM) approach (WOHLIN et al., 2012) as follows:

Analyze the literature on PLA and SAR for the purpose of characterization with respect to its relationship, evolution, support to PLA recovery, and research trends from the point of view of researchers and practitioners in the context of the SPL engineering field.

Thus, the contributions of this Chapter are (i) review the literature about the relationship between PLA and SAR, and its evolution over the years, (ii) investigate how existing approaches support PLA recovery, and (iii) identify research trends on PLA recovery.

3.2 RESEARCH PROCESS

The research design for this study was based on systematic literature review guidelines (KITCHENHAM; CHARTERS, 2007; BRERETON; BUDGEN; KITCHENHAM, 2016) to aim at a credible and fair evaluation of studies on PLA recovery. An important step is the protocol development. The protocol maps systematically the steps performed in the review and increases its rigor allowing study replication.

The review process, from planning to reporting, was carried out during twelve months by four software engineering researchers: one PhD student, and three researchers with expertise in SPL and architecture. All participants have experience in SPL projects in both industry and academia.

3.2.1 Research Questions (RQs)

Based on the study main goal, our review objective is to answer the following research questions:

- **RQ 1: How has the relationship between PLA and SAR evolved over the years?**

This research question aims to provide an overview of the research area. We seek to understand the relationship between SAR and PLA, and to investigate how this relationship evolved over the years. We verified the contribution type (Approach, Framework, Method, Process, Tool, and so on) and type of empirical research (Experiment, Case Study, Survey, and so on (EASTERBROOK, 2007)) performed in each study and how the research groups use them (Are the groups using the contributions from others?). We analyzed studies addressing the same issue – performed by the same authors in the majority of the cases – to verify its evolution over the years.

- **RQ 2: How does the existing solutions support PLA recovery?**

The goal of this question is to identify how the existing solutions, as reported in the literature, support PLA recovery. In this context, we identified the research type (Solution Proposal, Validation Research, Evaluation Research, Philosophical Papers, Opinion Papers, and Experience Papers (WIERINGA et al., 2005)) performed in the studies. We also verified if the studies adapted the existing SS solution proposal to recover PLA in the context of SPL projects. Moreover, we searched the solution proposal developed exclusively to recover PLAs. We also extended

the analysis of this RQ by performing an exploratory study, in which we used an existing clustering algorithm – commonly used to SAR in SS projects – to recover the architecture of SPL products.

- **RQ 3: What are the PLA recovery trends (according to SAR taxonomy axes)?**

By answering this question, we mapped some trends identified in the studies during the last few years. The objective of this question is to determine the SAR taxonomy axes – goals, inputs (architectural and non-architectural), outputs (visual, architecture, conformance, and analysis), techniques (quasi-manual, semi-automatic, and quasi-automatic), and processes (bottom-up, top-down, and hybrid) – proposed by Ducasse and Pollet (2009). We also extended the taxonomy by including elements focused on PLA recovery.

3.2.2 Search Strategy

The search strategy enables the inclusion of relevant studies in the search results. The search was based on (i) preliminary searches in key venues such as Software Product Line Conference (SPLC), European Conference on Software Architecture (ECSA), Working IEEE / IFIP Conference on Software Architecture (WICSA), International Conference on Software Engineering (ICSE), International Conference on Software Reuse, and so on. The lists of Journals and Conferences are detailed in Appendix A, (ii) trial searches using various combinations of search terms derived from the research questions, (iii) automatic search by executing search strings on search engines of electronic data sources, and (iv) the “snow-balling” process (BUDGEN et al., 2008), in which the identified studies references were analyzed.

When manually searching the venues, we considered title, keywords, and abstract. On the other hand, the search string for automatic search consisted of three parts: recovery AND software architecture AND software product line. The words were combined through logical OR to form a string, as can be seen on following search string:

(“recovery” OR “recover” OR “reconstruction” OR “reconstruct” OR “refactoring”) AND (“architecture” OR “architecting” OR “design” OR “designing”) AND (“product line” OR “product lines” OR “product-line” OR “product family” OR “product families” OR “SPL” OR “PLA” OR “SPLA”)

We changed the search string according to search features of electronic sources. For this reason, we used different search strings for different sources. However, these strings were semantically and logically equivalent.

3.2.3 Selection Criteria

In order to filter the studies and eliminate irrelevant ones (*e.g.* do not address the research questions), we adopted the following inclusion and exclusion criteria.

3.2.3.1 Inclusion: The study must explore SAR in SPL context, or present SAR solution proposals and how it supports PLA, or presents PLA recovery trends. Moreover, we only included studies written in English.

3.2.3.2 Exclusion: We excluded studies if their focus, or main focus, was not SAR or if they did not present topics related with reuse and PLA recovery. Studies available as abstracts or presentations were excluded.

3.2.4 Data Sources

We performed the search by adjusting the search strings and applied in each digital database, and all search strings were systematically checked by more than one author. The list of sources is the following: ACM Digital Library¹, Elsevier – Compendex², IEEE Computer Society Digital Library³, Science@Direct⁴, The DBLP Computer Science Bibliography⁵, and Scopus⁶.

Figure 3.1 presents the systematic mapping process and the number of papers identified at each stage.

3.2.5 Data Collection

Based on the selection criteria, we selected 35 studies. They have been read in details to extract the data that address each research question. We created an extraction form to collect data composed by the following information: author(s), year, title, source, research type, answer for each research question, and information to categorize the study.

During the extraction, we classified some studies in more than one category. Thus, primary studies can appear more than once. An extracted data record was kept in Excel spreadsheet for analysis. For each study, we collected the following fields: Author(s), Year, Title, Venue, Venue Type, Research Type, Contribution Type, Study address PLA, Study address SAR, Recovery Type, Tool Support, Empirical Study, and quality assessment answers. For each paper, data was extracted by one researcher and checked against the paper by another researcher. Disagreements were resolved by discussions between researchers or by consulting an additional researcher. More details can be seen in the study website⁷.

3.2.6 Data Analysis

We summarized data from primary studies to answer the research questions. Therefore, we reviewed data manually to perform a classification scheme that helped to categorize

¹<<http://dl.acm.org>>

²<<http://www.engineeringvillage.com/search/quick.url>>

³<<http://ieeexplore.ieee.org/>>

⁴<<http://www.sciencedirect.com/>>

⁵<<http://dblp.uni-trier.de>>

⁶<<http://www.scopus.com/>>

⁷<<https://goo.gl/l39HrF>>

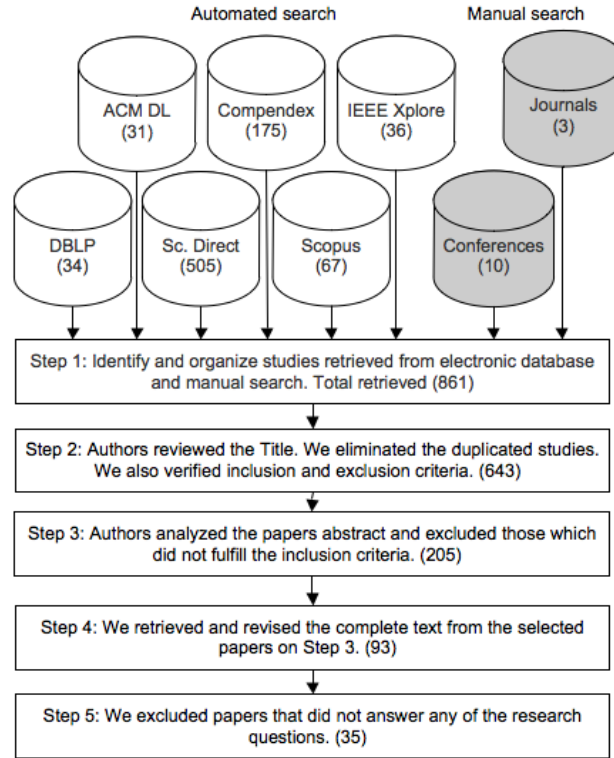


Figure 3.1 Paper selection flowchart

the results of this study in tabular form. In this way, we classified the studies according to contribution type (Approach, Framework, Method, Process, Taxonomy, and Tool) and empirical research (Experiment, Case study, Survey, Ethnography, and so on) (EAST-ERBROOK, 2007).

Moreover, we collected the research type facet defined by Wieringa et al. (2005). The categories are (i) *Validation Research* – techniques investigated are novel and have not yet been implemented in practice, (ii) *Evaluation Research* – techniques are implemented in practice and an evaluation of the technique is conducted, (iii) *Solution Proposal* – a solution for a problem is proposed, the solution can be either novel or a significant extension of an existing technique, (iv) *Philosophical Papers* – these papers sketch a new way of looking at existing things by structuring the field in form of a taxonomy or conceptual framework, (v) *Opinion Papers* – these papers express the researcher personal opinion, and (iv) *Experience Papers* – explain on what and how something has been done in practice. It has to be the personal experience of the author.

We identified the SAR taxonomy axes according to Ducasse and Pollet (2009). The goals verified if the SAR is the basis for redocumentation, reuse investigation, and migration to product lines, or coevolution of implementation and architecture.

There are three types of SAR processes: (i) Bottom-up processes start with the low-level knowledge to recover architecture, (ii) Top-Down processes start with the high-level knowledge to discover architecture by formulating conceptual hypotheses, and (iii) Hybrid

processes combine bottom-up with top-down processes.

Most SAR approaches are based on source code information and human expertise as input. However, some exploit other architectural or non-architectural information (DUCASSE; POLLET, 2009). On the other hand, the studies provide architectural views and information about the conformance of architecture and implementation as outputs.

SAR techniques are classified in quasi-manual (*i.e.* assisted techniques to manipulate knowledge), semi-automatic (*i.e.* automation of some aspects of SAR), and quasi-automatic (*i.e.* combination of concept, dominance, and cluster analysis techniques).

3.2.7 Quality Assessment

We defined the quality assessment (QA) to evaluate the credibility, completeness, and relevance of the studies based on a set of 11 quality criteria. Six of them were adapted from existing studies, and the remaining five questions were proposed to evaluate specific questions regarding this literature review. Table 3.1 presents the assessment instrument. Criteria A, C, G, H, I, and J were adopted from recent SLRs (DYBÅ; DINGSØYR, 2008; MAHDAVI-HEZAVEHI; GALSTER; AVGERIOU, 2013; DERMEVAL et al., 2014; DING et al., 2014). On the other hand, criteria B, D, E, F, and L were proposed in this work.

Table 3.1 Study Quality Assessment Criteria

ID	Questions	Possible Answers
A	Is there a rationale for why the study was undertaken?	Yes = 1.0, No = 0.0, Partially = 0.5
B	Is the paper an extension or complete previous research?	Yes = 1.0, No = 0.0
C	Is there a clear statement of the aims of the research?	Yes = 1.0, No = 0.0, Partially = 0.5
D	Does the study reuse an existing SAR technique?	Yes = 1.0, No = 0.0
E	Does the study clearly describe the PLA recovery?	Yes = 1.0, No = 0.0, Partially = 0.5
F	Does the study provide guidelines to support PLA recovery?	Yes = 1.0, No = 0.0, Partially = 0.5
G	Is the study supported by a tool?	Yes = 1.0, No = 0.0
H	Was the study empirically evaluated?	Yes = 1.0, No = 0.0
I	Is there a discussion about the results of the study?	Yes = 1.0, No = 0.0, Partially = 0.5
J	Are the limitations of this study explicitly discussed?	Yes = 1.0, No = 0.0, Partially = 0.5
L	Does the study describe the variability identification in arch. level?	Yes = 1.0, No = 0.0, Partially = 0.5

We determined the questions B, D, G, and H scores using a two-grade scale (**Yes/No**). The study received 1.0 point to **Yes** answer and 0.0 point to **No** answer. The questions A, C, E, F, I, J, and L used a three-grade scale, allowing a third answer (0.5) in case the contribution was not so strong.

We computed the study quality score by summing its answers scores to the questions. The authors assessed each paper and discussed the discrepancies. They also reevaluated the studies in case of non-agreement.

3.3 OUTCOMES

We used the extracted data to answer the research questions. Moreover, we describe the main findings and group the results according to each research question. The following tables and figures contain the studies number and classification. First, we give an overview

of the identified studies and extracted information. Then, we answered the research questions by analyzing the data.

3.3.1 Characteristics of the studies

The 35 selected studies encompass the years 1998 through the first semester of 2017. They were gathered from 24 conferences, 1 workshop, and 6 journals. The remaining 4 studies were published as one book chapter, one technical report, and two Ph.D. Theses. The list of selected studies can be found in Appendix A.

As expected, the greater amount of studies in a single vehicle was found in SPLC (4 studies), considered the most representative conference for the SPL engineering area, followed by WICSA (3 studies), International Conference on Software Engineering (3 studies), European Conference on Software Maintenance and Reengineering (2 studies), and Journal of Systems and Software (2 studies).

Figure 3.2 shows the distribution of studies according to their publication years. Such a distribution gives us the initial impression that most relevant studies in the field were found in recent publications, *i.e.*, as of the year 2011, 2014, and 2016.

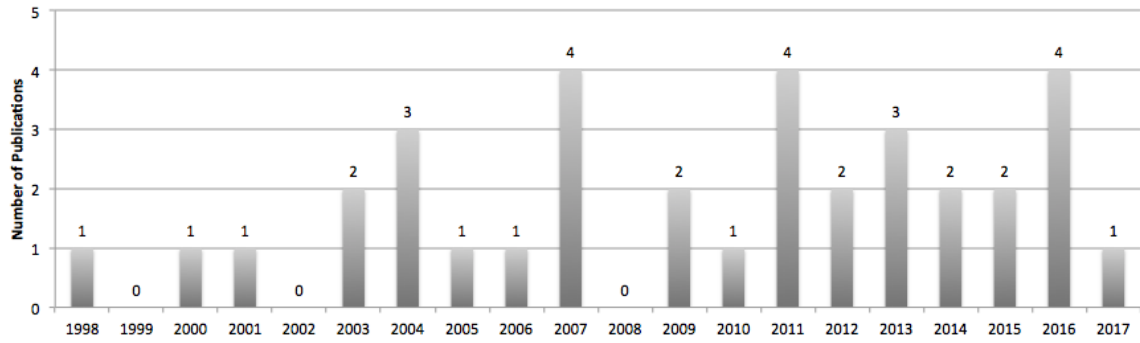


Figure 3.2 Temporal view of the studies

We notice a trend curve in the data, showing an increasing attention on the use of scientifically rigorous evaluation methods as a means to assess and make explicit the value of the proposed approaches for the SAR field.

3.3.1.1 Quality Assessment Results

The quality assessment improved the accuracy of data extraction results. This evaluation helped to determine the validity of the inferences conducted and verify the credibility and coherence of the result synthesis.

Table 3.2 right side presents the QA results according to the assessment questions described in Table 3.1. Only four studies presented scores less than 40% (S1, S8, S18, and S22). These 11 criteria provided a confidence measure to verify if a particular study findings provide a valuable contribution to this review.

We identified that three studies published in journals presented the highest quality scores of the review, S15 (91%), S34 (91%), and S17 (86%). A possible reason is that journals provide a combination of rigor and increase space for discussing the research

topics in depth. We identified some studies published in conferences with high-quality scores – S29 (82%) and S27 (77%).

3.3.2 Results

Table 3.2 presents the list of 35 reviewed studies (33 primary studies and 2 secondary studies – S11 and S16). Based on the data, we can consider PLA recovery as a recent research area. For this reason, the findings discussed here should be considered as initial evidence or tendencies.

Table 3.2 Classification of Reviewed Studies

ID	P.	R.	Pro.	E.R.	A	B	C	D	E	F	G	H	I	J	L	Tot.	Qual.
S1	•	•	hyb.	C.S.	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.5	0.0	0.0	2.5	23%
S2	•	•	hyb.	C.S.	1.0	1.0	1.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	5.0	45%
S3	•	⊖	hyb.	C.S.	1.0	0.0	0.5	0.0	0.5	0.0	1.0	1.0	1.0	0.0	0.0	5.0	45%
S4	⊖	•	B.U.	-	1.0	0.0	1.0	0.0	0.5	1.0	1.0	0.0	1.0	0.0	0.0	5.5	50%
S5	•	•	hyb.	-	1.0	0.0	1.0	1.0	0.5	0.0	1.0	0.0	1.0	1.0	1.0	7.5	68%
S6	-	•	hyb.	C.S.	1.0	0.0	0.5	1.0	0.0	0.0	1.0	1.0	0.5	0.0	0.0	5.0	45%
S7	⊖	•	hyb.	C.S.	1.0	0.0	1.0	0.0	0.5	0.0	1.0	1.0	1.0	1.0	0.0	6.5	59%
S8	⊖	⊖	T.D.	-	0.0	0.0	1.0	0.0	0.5	0.0	0.0	0.0	1.0	1.0	0.0	3.5	32%
S9	•	•	B.U.	C.S.	1.0	0.0	0.5	0.0	0.5	0.0	0.0	1.0	0.5	1.0	0.0	4.5	41%
S10	-	•	T.D.	C.S.	0.0	1.0	0.5	1.0	0.0	0.0	1.0	1.0	0.5	0.0	0.0	5.0	45%
S11	⊖	•	All	sur.	0.5	0.0	1.0	1.0	0.5	0.0	1.0	1.0	1.0	0.0	0.0	6.0	55%
S12	⊖	•	hyb.	C.S.	0.5	0.0	1.0	1.0	0.5	0.0	0.0	1.0	0.5	0.0	1.0	5.5	50%
S13	⊖	•	hyb.	exp.	0.5	1.0	1.0	1.0	0.5	0.0	1.0	1.0	1.0	0.5	1.0	8.5	77%
S14	•	•	hyb.	C.S.	1.0	0.0	1.0	1.0	0.5	0.0	1.0	1.0	1.0	1.0	1.0	8.5	77%
S15	•	•	hyb.	C.S.	1.0	1.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0	10.0	91%
S16	⊖	•	All	sur.	1.0	1.0	1.0	1.0	0.5	0.0	1.0	1.0	1.0	0.0	0.0	7.5	68%
S17	⊖	•	hyb.	exp.	1.0	1.0	1.0	1.0	0.5	0.0	1.0	1.0	1.0	1.0	1.0	9.5	86%
S18	•	-	-	-	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	4.0	36%
S19	⊖	⊖	hyb.	C.S.	1.0	0.0	0.5	0.0	0.5	0.0	1.0	1.0	1.0	1.0	1.0	7.0	64%
S20	•	-	-	-	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	5.0	45%
S21	⊖	•	T.D.	C.S.	1.0	0.0	1.0	0.0	0.0	0.0	1.0	1.0	1.0	0.0	1.0	6.0	55%
S22	-	•	B.U.	-	1.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	4.0	36%
S23	•	•	B.U.	C.S.	1.0	1.0	1.0	0.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	8.0	73%
S24	•	-	-	C.S.	1.0	1.0	1.0	0.0	1.0	0.0	0.0	1.0	1.0	0.0	1.0	7.0	64%
S25	-	•	B.U.	sur.	1.0	0.0	1.0	1.0	0.0	0.0	1.0	1.0	1.0	1.0	0.0	7.0	64%
S26	•	⊖	B.U.	C.S.	0.5	0.0	1.0	0.0	1.0	0.0	1.0	1.0	1.0	0.0	0.0	5.5	50%
S27	•	•	hyb.	C.S.	1.0	0.0	1.0	1.0	0.5	0.0	1.0	1.0	1.0	1.0	1.0	8.5	77%
S28	-	•	B.U.	-	1.0	1.0	0.5	1.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	5.5	50%
S29	•	•	B.U.	exp.	1.0	0.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0	9.0	82%
S30	-	•	B.U.	exp.	1.0	1.0	1.0	1.0	0.0	0.0	1.0	1.0	1.0	1.0	0.0	8.0	73%
S31	-	•	B.U.	C.S.	1.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	1.0	6.0	55%
S32	-	•	B.U.	exp.	1.0	0.0	1.0	1.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	8.0	73%
S33	-	•	B.U.	-	1.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	1.0	0.5	0.0	4.5	41%
S34	•	•	B.U.	exp.	1.0	1.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0	10.0	91%
S35	-	•	B.U.	exp.	1.0	0.0	1.0	0.0	0.0	0.0	1.0	1.0	1.0	1.0	0.0	6.0	55%

Legend: [•] Characteristic clearly addressed by the study, [⊖] The study encourages the use of such characteristic, but do not provide any implementation (*e.g.*, it states an external tool is used, but no detail is provided), [-] Characteristic not mentioned in the study, [P.] PLA, [R.] Recovery, [Pro.] Process, [hyb.] Hybrid, [B.U.] Bottom-up, [T.D.] Top-Down, [E.R.] Empirical Research, [C.S.] Case Study, [sur.] Survey, [exp.] Experiment, [Tot.] Total Score, and [Qual.] Quality Score. Items A to L are described in Table 3.1

Most of the studies (74%) discuss PLA, SAR, and their relationship. Although some of them (26%) did not address PLA, these studies were included because (i) they considered at least one SPL aspect such as software reuse or variability, or (ii) they described research trends adaptable to PLA.

A total of sixteen studies proposed the development of tools to support SAR in the context of SPL projects, four of them use external tools (developed by third parties), but no detail is provided, and ten studies did not mention any tool support. Without proper tool support, it may not be feasible in practice to carry out SAR.

Regarding empirical assessment performed on the basis of the proposed SAR approaches, 54% of the studies presented case studies, 14% presented experiments, 9% presented surveys, and 23% did not present empirical studies. The numbers indicate the need of more empirical evaluation.

In combination with Table 3.2, we present a timeline in Figure 3.3 that shows additional information. It shows papers distribution over the years until 2017. Moreover, Figure 3.3 shows how many papers were found in journals, conferences, book chapter, technical reports, PhD thesis, or workshops. The lines connecting the papers indicate when a paper was extended to a new publication.

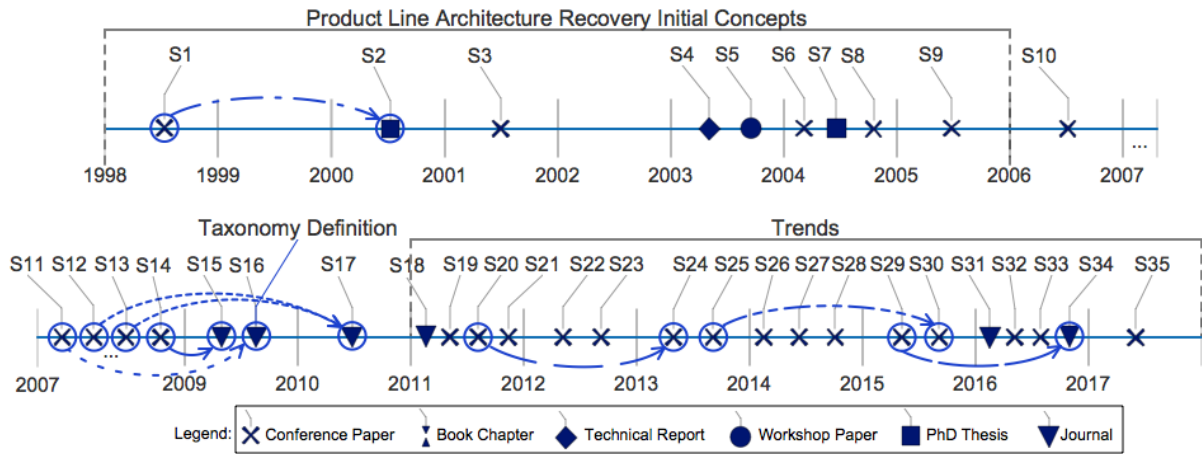


Figure 3.3 Timeline

3.3.3 RQ 1: How does the relationship between PLA and SAR evolve over the years?

To answer this question, we continue the timeline analysis. As can be seen in Figure 3.3, the timeline summarizes the studies evolution over the years. We identified three phases: (i) definition of basic concepts – initial studies considering PLA and SAR; (ii) field consolidation – first journals addressing PLA and SAR; and, (iii) appearance of new research trends.

Eixelsberger performed the first studies combining PLA and SAR (S1 and S2). The author started investigating SAR of embedded software (EIXELSBERGER et al., 1997) and architectural structure (EIXELSBERGER et al., 1998). Later, the research evolved to architecture recovery of product lines.

S5 proposed the architecture recovery of individual systems from the same domain and compared them to allow the SPL Design Reference Architecture (DSSA) creation. The

study presented techniques along with tools integrated into the reconstruction process to reduce the manual effort. Similarly, S9 recovered the conceptual architecture from legacy applications of the same domain. The authors identified services based on the extracted object relationship diagram.

Further, two studies (S14 and S15) extended the reflexion method to compare product implementations to the product architecture and to compare product architectures to the SPL architecture. The studies used clustering algorithms to consolidate software variants into product lines, assuming that implementations and architectures of the variants are similar.

From 2011 and on, studies presented trends (further discussed in RQ 3) and demonstrated the maturity on the development of SAR solution proposals. The studies are mostly focused on recovering the variability at the requirement level (S2, S5, S9, S21, S23). Few works aim at PLA recovery focusing the variability at the architectural level (S9, S14, S15). Only two studies (S29 and S34) proposed fully automated recovery approach, but it recovered the PLA based on different versions of the same SPL project – *e.g.* Health Watcher (GREENWOOD et al., 2007) and MobileMedia (FIGUEIREDO et al., 2008).

3.3.4 RQ 2: How does the existing solution proposal support PLA recovery?

Table 3.3 shows the amount of studies per research type. Most studies (24 of 35, 73%) presented solution proposal. Moreover, 11% of the studies performed Evaluation Research, and 11% are Experience Papers. Two studies are philosophical papers (S11 and S16) providing the SAR taxonomy, and only one study (S24) performed Validation Research.

Table 3.3 Number of studies per research type

Classes	Studies	Count
Solution Proposal	S1–S8, S12–S15, S17–S19, S21–S23, S26, S28, S29, S31–S35	26
Experience Papers	S9, S10, S20, S27	4
Evaluation Research	S25, S30, S32, S34	4
Philosophical Papers	S11, S16	2
Validation Research	S24	1

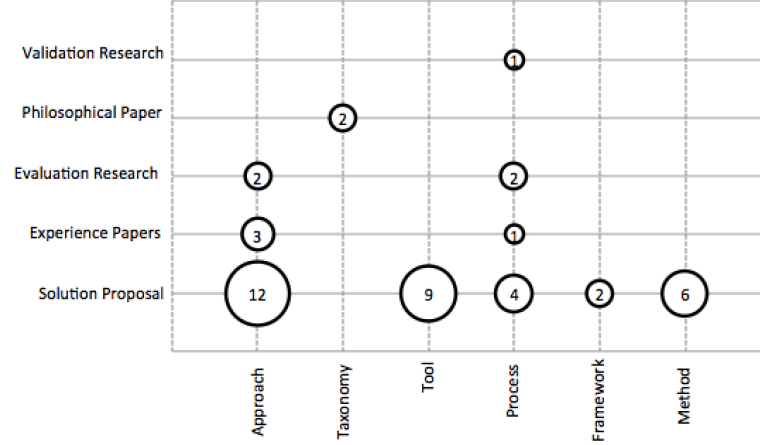
To provide a detailed view regarding the solution proposals, Table 3.4 organizes only the studies addressing solution proposal per research contribution. In this way, the studies presented twelve approaches, nine tools, six methods, four processes, and two frameworks. These numbers are reflected in Figure 3.4.

Figure 3.4 shows the relationship between research type and research contribution. We identified three experience papers evaluating approaches and one evaluating process. Moreover, two studies provided evaluation research for approaches and two for processes. The two philosophical papers (S11 and S16) provided a taxonomy for SAR and only one validation research evaluated a process.

Studies S1, and S2 presented a set of methods that use information from members of a family of legacy software systems to recovery the architectural properties in the

Table 3.4 Solution proposal per research contribution

Classes	Studies	Count
Approach	S6, S8, S12, S13, S17, S26, S29, S31–S35	12
Tool	S4, S7, S14, S15, S17, S21, S23, S28, S33	9
Method	S2, S3, S7, S14, S15, S19	6
Process	S4, S5, S18, S21	4
Framework	S1, S22	2

**Figure 3.4** Research types per research contribution

systems and build architectural descriptions. S3 proposed a bottom-up approach to recover architectural representations of existing systems and a top-down approach to map known architecture architectural styles and attributes onto the recovered architecture.

Moreover, S4 describes the process of architecture reconstruction using the Architecture Reconstruction and Mining (*ARMIN*) tool. The representation can be used as a way for identifying components for reuse or for establishing an architecture-based SPL. S5 combines solution proposals (techniques and tools) to analyze related systems, determine common assets, and integrate these assets into the design of a reference architecture. On the other hand, S7 proposes a method, called *NIMETA*, for architecture reconstruction based on the recovery of the architecturally significant views.

Not always tools implement SAR techniques. For example, S12, S13, and S17 presented *ArchMine*, an architecture recovery approach based on dynamic analysis and data mining. S14 and S15 described a method and supporting tools to compare software variants at the architectural level. The method consists of the specification of the module view and the mapping of implementation components onto the module view.

Furthermore, S21 presented a tool-supported approach to reverse engineer architectural feature models. Similarly, S23 proposed a history-sensitive heuristics (supported by a tool *RecFeat*) for the recovery of features in code of degenerate program families.

S22 proposed to explore the use of machine learning techniques to automatically recover software architecture from software artifacts, S28 presented *ArchViz* a tool that partially automate the analysis of recovered architectures. S29 and S34 proposed an

approach to reverse engineer the architecture of a set of product variants. The study relies on Formal Concept Analysis (FCA) to analyze the variability.

3.3.5 RQ 3: What are the PLA recovery trends?

To identify possible research trends in the context of PLA recovery, Figure 3.5 presents the results for data extraction. As explained previously, we organized the information according to SAR taxonomy axes defined by Ducasse and Pollet (2009). Again, one study can use more than one data source type.

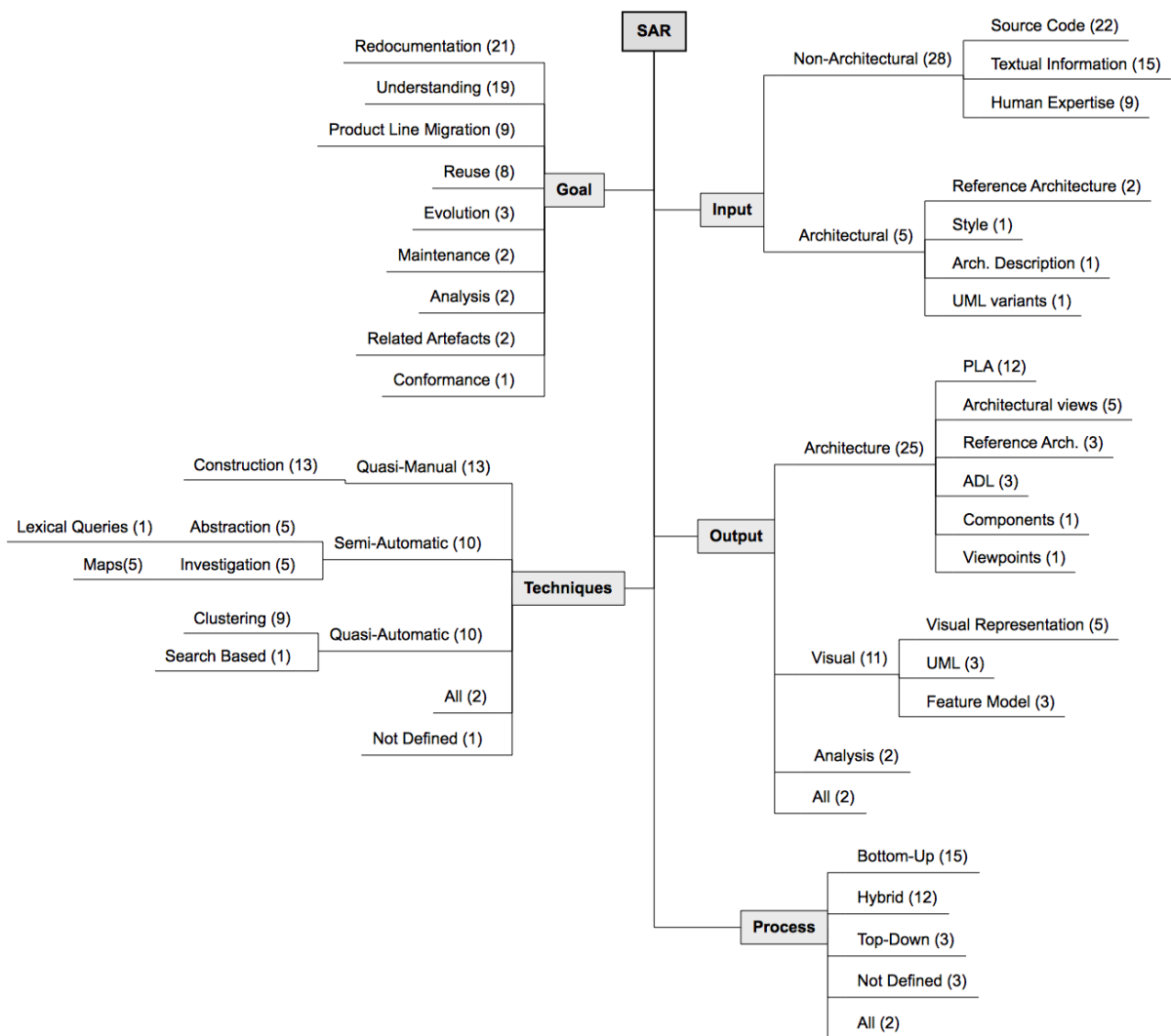


Figure 3.5 Number of studies per SAR taxonomy axes

The numbers in the parenthesis represent the total of studies per each category. As can be seen in Figure 3.5, for instance, there are 22 studies using Source code, 15 studies using Textual information, and 9 studies using human expertise as non-architectural inputs. Regarding SAR process, 43% of the studies defined bottom-up processes, 9%

top-down, 34% hybrid, two studies considered these three types, and three studies did not address any of them.

Regarding the SAR Goal axis, most of the studies focused on Redocumentation (60%) and Understanding (54%). On the other hand, 26% of the studies (9 out of 35) addressed Product Line Migration and 23% on Reuse. Moreover, we identified 13 studies addressing Quasi-Manual techniques, 10 Semi-Automatic techniques, and 10 studies focusing on Semi-Automatic techniques. Most of the studies (25 out of 35) provided the architecture as an output from the SAR – 12 of them focused on the PLA.

Based on this previous analysis, we analyzed PLA recovery trends and SAR used to recovery single systems architecture adaptable to PLA recovery. Primarily, the research community demands more studies and evidence presenting empirical studies such as performed in S10, S25, and S30. Most of studies consists of solution proposals (73%), next step suggests the development of studies describing experiments, surveys, and empirical evaluation.

S22 indicates a trend towards SAR automation. The study goal is to develop techniques allowing automatic architecture recovery from source code. The study proposes the use of machine learning techniques to automatically recover software architecture. On the other hand, S23 proposes history-sensitive heuristic for features recovery in source of degenerating program families.

Another trend is related to design PLA based on reference architectures (S20 and S24). In S24, the authors present a process, named *ProSA-RA2PLA*, that uses reference architectures for building PLAs systematically.

There is a trend related to SAR tools development (S23 and S28). New tools are necessary to support solution proposal (*e.g.* approaches, frameworks, heuristic, methods, etc) in PLA recovery. One brief example might clarify this concept, it is feasible to use SAR tool to recover reference architectures and enable PLA construction (as proposed in S20 and S24).

S27 characterizes a research trend related to architectural bad smells in the context of PLA through an exploratory study. Recently, S29 and S34 presented research trends related to identify variability and dependencies among architectural variants at the architectural level.

3.4 DISCUSSION

In the following, we provide a summary of the main findings, limitations to the review, and threats to validity.

3.4.1 Main Findings

The goal of this review is to identify the studies to define PLA recovery by analyzing the relationship between PLA and SAR over the years, verify whether the existing solution proposal are being used in PLA recovery, and identify the PLA recovery trends.

While most of the identified studies are solution proposal, few studies provide details on empirical assessment they conducted to validate whether their proposal delivered

interesting results. Figure 3.6 shows the relationship among contribution type, research type, and research questions. Although generalizing the findings may not be feasible at this point, the gathered evidence indicates the lack of empirical evaluation, and according to Table 3.3, a small number of opinion or theoretical studies investigate the area. We identified only one study providing guidelines to support the PLA recovery (S4).

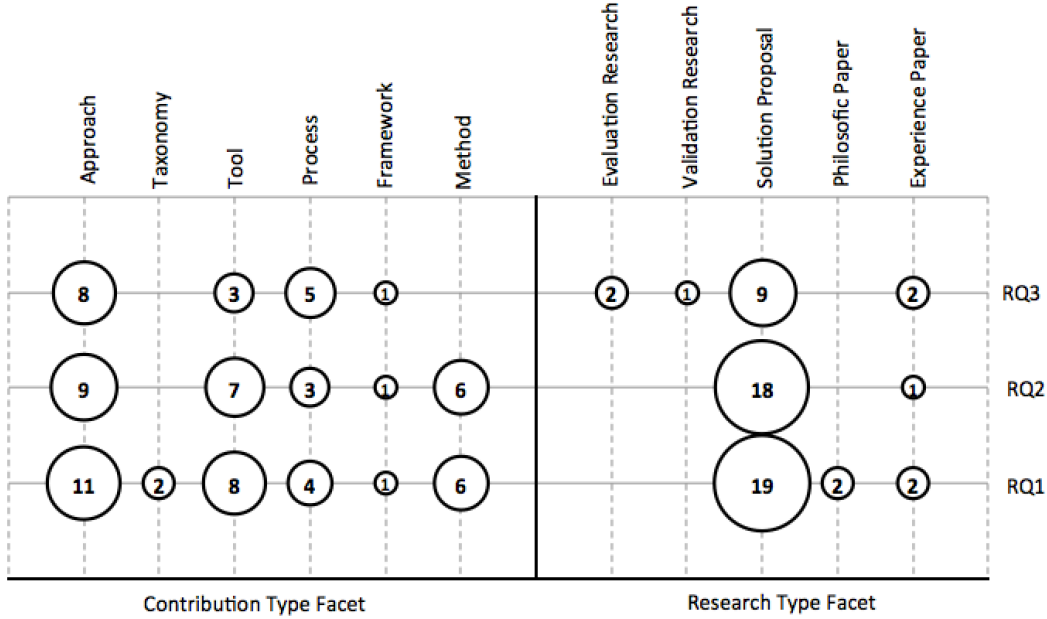


Figure 3.6 Contribution type versus research type versus research questions

3.4.1.1 Answering RQ1

Figure 3.7 shows the seven Evolution Lines (EL) we identified during the analysis. We found that some studies evolved from conference papers to journals (*e.g.* EL1 – S11 evolved to S16, EL3 – S12 and S13 evolved to S17, and EL7 – S29 evolved to S34). Probably because journals provided more space for the authors to describe their work in details. In other words, studies discussed their research topic in depth. In S20, the authors explored the use of reference architectures in the development of SPL artifacts. Further, the authors evolved the research to S24, presenting a process to systematize the use of existing reference architectures to build PLA.

Moreover, we observed that most of the ELs was performed by the same research group. In other words, the groups did not use the solution proposal from other groups, except for EL2 and EL6. In the former, the researchers analyzed a set of studies to define SAR taxonomy. In the latter, the research group performed a comparative analysis of studies.

During our review, we also identified the evolution of SAR techniques (S25 and S30), processes (S18 and S24), and tools (S28 and S29). They became more complex in solving specific issues of SAR field – for instance, by improving the precision and recall of the clustering algorithms. Moreover, we found tools developed to support heuristics (S23),

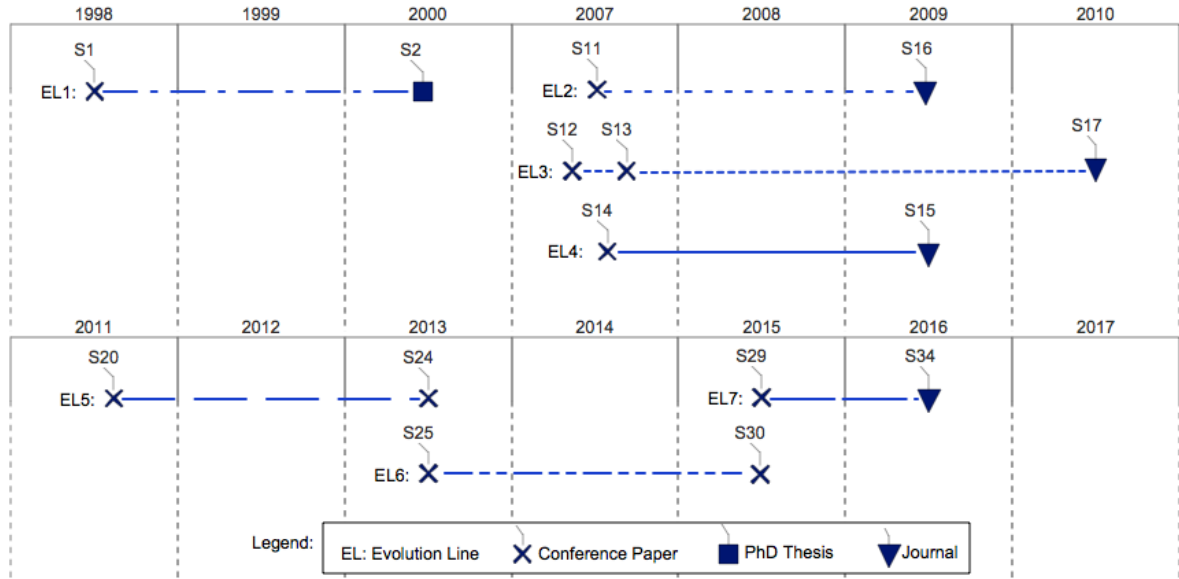


Figure 3.7 Evolution lines identified

approaches (S17 and S19), and frameworks (S22).

Figure 3.8 shows the number of empirical studies over the years. We identified that the number of Case Studies remains constant since 2011. The main likely reason is the high number of solution proposals, researchers investigate their solutions and apply them in Case Studies. Moreover, we observed that since 2015 there is at least one study performing controlled experiments. This evidence can indicate the concern with more rigorous evaluation.

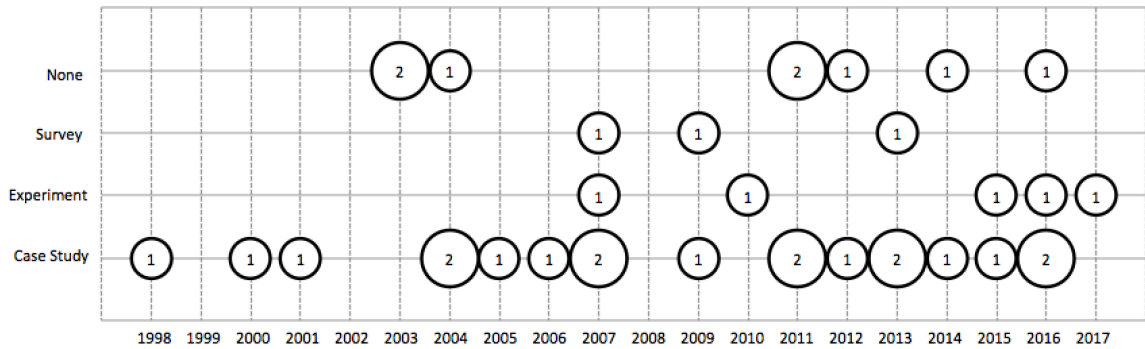


Figure 3.8 Empirical studies over the years

Even with the evolution in this aspect, there is no standardization regarding inputs and outputs compromising the comparison among the existing tools and techniques. For this reason, we identified a small number of studies carrying out controlled experiments. Moreover, the majority of the research groups work independently without reusing the solution proposed by other groups – as we raised in the previous EL analysis. We verified the reuse of tools in few studies (S25 and S30); probably because the research groups

prefer to develop tools focused on their context which is reasonable when working with specific problems.

The SAR field supports PLA recovery by providing solution proposal adaptable to SPL context (e. g. adapted processes, tools, approaches, frameworks, and so on). Few works focused on fully automated recovery of variability at the architectural level (S29 and S34).

3.4.1.2 Answering RQ2

Most of the solution proposals are used to recover the architecture from legacy systems source code to create the SPL reference architecture (S1, S2, S3, S4, S5, S12, S13, S17, S22). In other words, these solution proposals support the migration from legacy code into SPL and five of them (S3, S4, S12, S13, and S17) presented architecture mining techniques to migrate legacy systems into product line.

As Figure 3.9 shows, new approaches, tools, and techniques were developed to address specific problems such as recovering architectural variability (S22, S29, and S34), recovery of feature models (S21 and S23), and static architecture variants reconstruction, allowing module view reconstruction (S14 and S15).

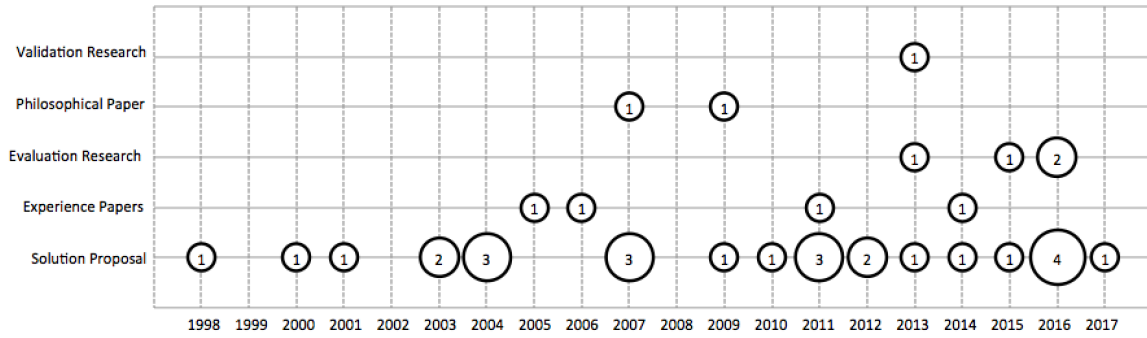


Figure 3.9 Research type over the years

Since 2009, there is at least one solution proposal per year. As a matter of fact, because of the appearance of new solutions over the few years, we believe that SAR solution proposals used to recover single systems architecture could be used to recover software architecture of SPL projects. For example, it is feasible to adapt the architecture recovery techniques presented in S25 and S30 to solve PLA recovery challenges.

Another example, S28 developed *ArchViz*, a tool to compare recovery architectures that also could be used to support PLA.

Figure 3.10 shows the analysis of SAR goals combined with research type and empirical evaluation. As can be seen, the majority of solution proposal focused on documentation and understanding. In addition, most of the studies performed case studies.

3.4.1.3 Answering RQ3

We identified the following research trends:

- T1: Architectural variability recovery (S22, S29, and S34);

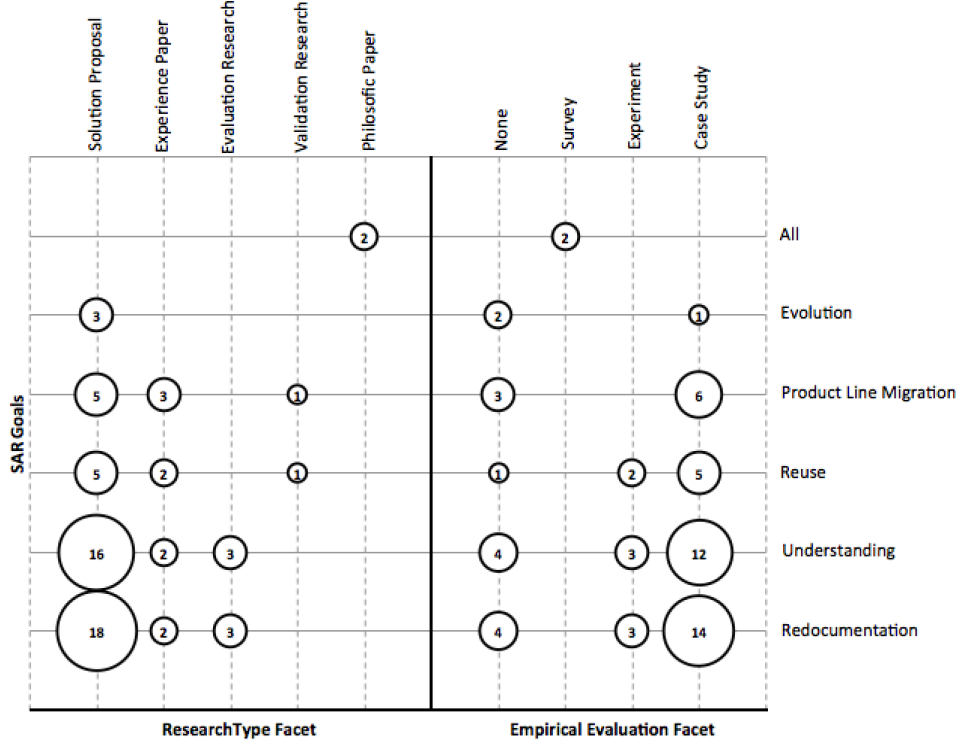


Figure 3.10 SAR goals vs. research type vs. empirical

- T2: Development of empirical studies (S20, S24, S25, S27, and S30);
- T3: PLA reconstruction based on reference architecture (S20 and S24);
- T4: Web-based tools implementation to support PLA recovery (S21, S23, S28, and S33);
- T5: PLA recovery to identify architectural bad smells in the SPL context (S27);
- T6: Search based techniques to recover PLA (S35).

Furthermore, the appearance of new development trends such as Cloud Computing (CC) (ZHANG et al., 2010; MOLLAH; ISLAM; ISLAM, 2012) enables the development of new solutions in the PLA recovery research field. As earlier mentioned in this chapter, there is an opportunity of extending these new technologies to solve some PLA recovery challenges. Because CC eliminates the platform dependency, the development of web-based tools (S28) could solve the SAR input and output standardization problem.

We identified that since 2012 the studies focused on bottom-up SAR process (see Figure 3.11). Evidence indicates that it is because the academia and industry focused on the development of new tools that allow the information extraction from source code.

Figure 3.12 presents the analysis of three SAR axes, SAR goals, SAR processes, and SAR techniques. Most of the studies focused on bottom-up processes with the goals of redocumentation and understanding through the use of Semi-Automatic techniques.

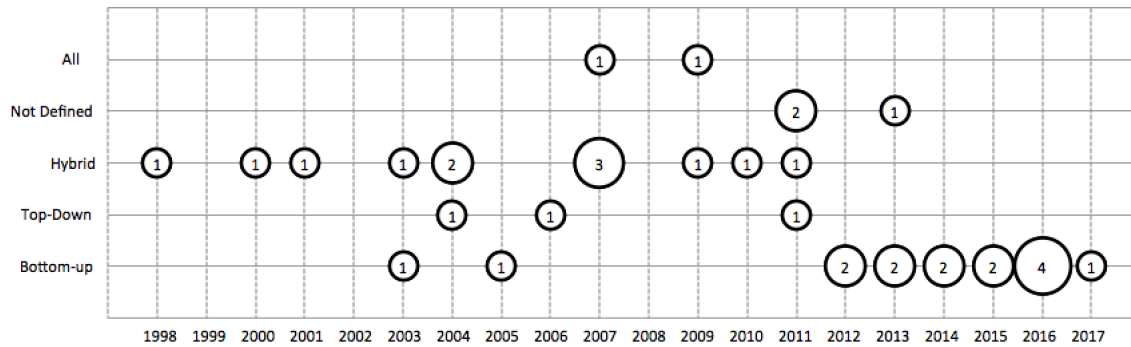


Figure 3.11 SAR processes over the years

Combined with the information from Figure 3.11, we confirmed the trend line towards of bottom-up processes.

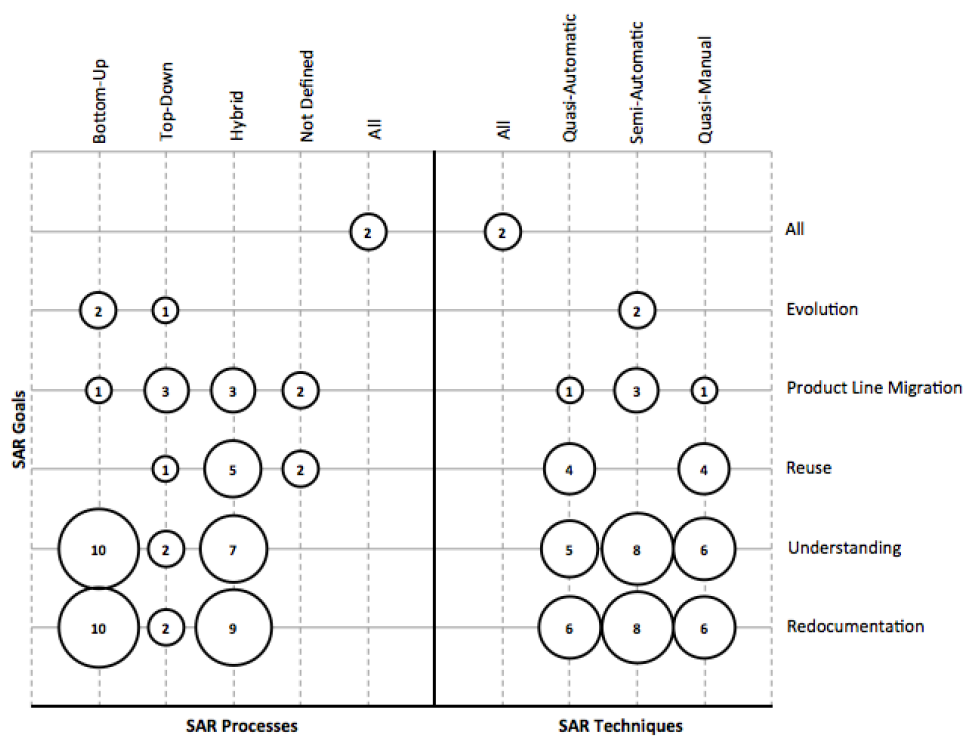


Figure 3.12 SAR goals vs. SAR processes vs. SAR techniques

There is a number of studies that did not explicitly answer the research questions. Table 3.5 presents the mapping between the studies and the research question. Even with some studies addressing the research questions, more evidence is required to consolidate the research field on PLA recovery. This topic is still an emerging research area and further investigation is necessary.

Table 3.5 Number of studies per research question

RQ	Studies	Count
RQ1	S1–S9, S11–S17, S19, S21, S23, S27, S29, S32–S35	25
RQ2	S1–S5, S7, S8, S10, S12–S15, S17, S19, S21, S23, S26, S29, S31, S34	20
RQ3	S18, S20–S32, S34, S35	16

3.4.2 Limitations of the Review and Threats to Validity

During automatic search, our main objective was to ensure the selected papers completeness. As mentioned, we searched a number of key venues. We also excluded irrelevant papers to reduce the researchers bias that affect the paper selection process. In this sense, there are some threats to the validity of our mapping, which we briefly describe along with the mitigation strategy for each threat.

- *Publication bias*: We cannot guarantee that all relevant primary studies were selected. It is possible that some relevant studies were not chosen during the search process. We mitigated this threat as much as possible, by following references in the primary studies.
- *Research questions*: The research questions we defined cannot provide complete coverage of the SAR field. We considered this as a threat, however, we discussed with researchers from our group and with an expert in the area to validate the questions.
- *Search conducted*: Although digital databases have incompatible search rules, we adapt our search strings for each digital database.

3.5 RELATED WORK

Pollet et al. (2007) presented a process-oriented software architecture reconstruction taxonomy. The authors proposed a classification based on the life time of SAR approaches. SMS, in 2009, Ducasse and Pollet (2009) extended the research. The authors analyzed briefly some aspects of SAR in the SPL context.

For example, they affirm that reconstructed architecture is the basis for reuse investigation and migration for product lines. Due to its relevance and because they synthesized information from previously published studies, in our review, we considered these studies as secondary studies.

Souza-Filho et al. (2008) presented a systematic review on domain design approaches to understand and summarize empirical evidence about their activities, identifying directions, strengths, and weaknesses.

Although, these studies provide relevant evidence, they did not consider PLA recovery deeply. Thus, in contrast, we aim to review the PLA recovery state-of-the-art and its evolution over the years, analyze the relationship between SAR and PLA, and identify SAR trends adaptable to PLA recovery. These research areas can work together and benefit from each other.

Sinkala, Blom and Herold (2018) performed a mapping study of SAR for SPLs. The authors interested in the architectural degradation in the context of systems developed with the clone and own strategy. Although the similarities with our mapping, only 22% of the identified studies overlap because they focused on the SPL evolution and migration.

3.6 CHAPTER SUMMARY

In this Chapter, we presented a SMS to gather evidence for understanding how PLA relates to SAR.

Regarding the mapping, the selected studies fell into three thematic groups: understand the relationship between PLA and SAR, characterize how the existing solution proposals support PLA, and identify the PLA recovery trends.

The PLA and SAR relationship evolved from the definition of basic concepts to provide solution proposals that solve specific problems. The solution proposals majority are used to recover the SPL reference architecture based on legacy systems. Only few studies address empirical evaluation such as experiments, surveys, mixed-methods, and so on. In this context, more empirical research is still necessary. There is a clear need to establish standardization for future field studies.

The mapping summarized the information providing a list of primary studies and identified initial evidence for understanding the relationship between PLA and SAR.

Energy and persistence conquer all things. – Benjamin Franklin

AN APPROACH FOR RECOVERING PLA FROM SOURCE CODE OF VARIANTS

This Chapter presents our approach called Software Architecture Variability Recovery (**SAVaR**) to recovering architectural variability from variants' source code. It supports the identification of the minimal subset of cross-product architectural information that results in a PLA. **SAVaR** includes a threshold technique that reduces the explosion of variability information in the PLA representation.

4.1 OVERVIEW

The aim of **SAVaR** is to support PLA recovery (and the variability identification) by systematically using bottom-up SAR tools and techniques in the SPL context through the use of guidelines to support **SAVaR**.

We used the *Software and Systems Process Engineering Meta-model (SPEM)*¹ to describe **SAVaR**. The following roles are involved in **SAVaR**: Recoverer, SPL Architect, and SPL Developer. The recoverer follows the steps described in the guidelines. Both SPL Architect and Developer could play such a role. The SPL architect understands and verifies the recovered PLA. Moreover, the SPL developer is responsible for verifying the recovered information according to the SPL implementation and checking the relationship between the SPL source code and the PLA.

Figure 4.1 presents an overview of the **SAVaR**. This representation allows the application of **SAVaR** in different contexts of PLA recovery such as we describe in Section 4.4.

SAVaR receives as inputs: mainly the SPL or variants' source code and the guidelines to support **SAVaR**. Examples of input candidates are SPL products, systems from the same domain, projects implemented using clone-and-own strategy, etc. The main phases are: ① SPL information collection, ② information extraction, ③ PLA recovery with variability identification, and ④ PLA presentation. **SAVaR** produces four main outputs: Recovered PLA, reports, collected metrics and DSM.

¹<http://omg.org/spec/SPEM/2.0/>

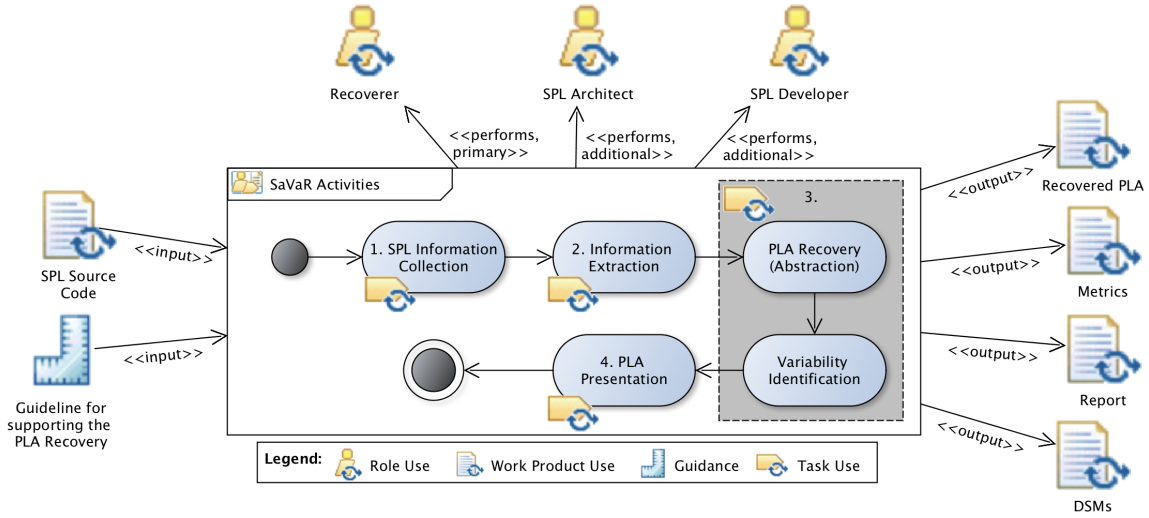


Figure 4.1 SAVaR Summary

Figure 4.2 presents the application of SAVaR in details. The SPL information collection ① is organized in the following steps. In step (a), the recoverer downloads the SPL project from the repository and analyzes the source code and other available assets (step (b)). Step (c) demands the instantiation of at least two variants and only happens with negative answers from the decisions **D1** and **D2**. When the SPL is implemented using `#ifdef` directives (Decision **D1**) or the variants already exist (Decision **D2**), the recoverer skips the step (c).

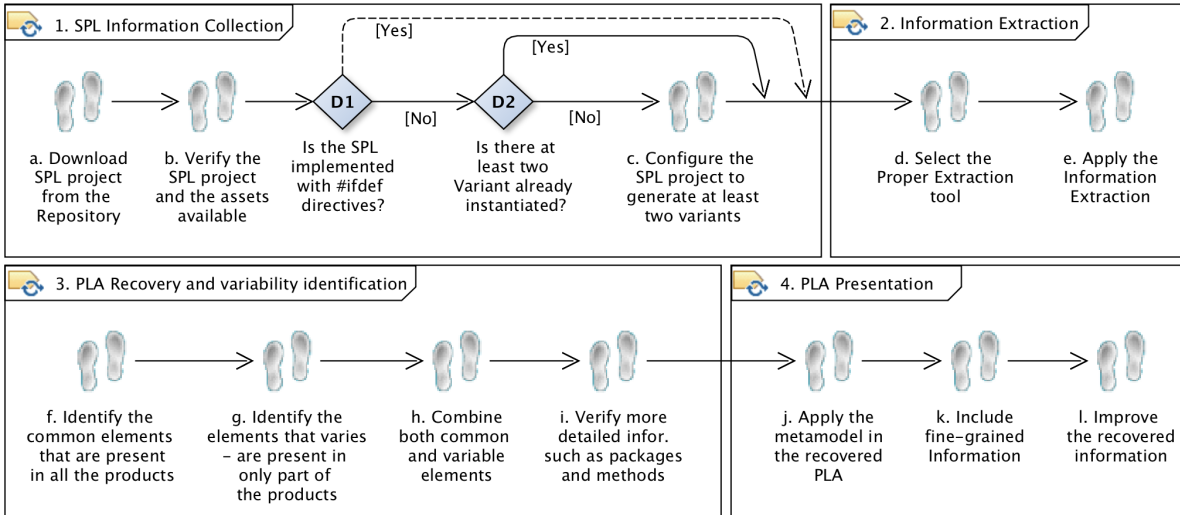


Figure 4.2 SAVaR Operation phase in details

The SPL information extraction ② is organized in the following steps. The recoverer selects the proper extraction information tool step (d) and performs the information extraction in step (e). The next phase, PLA recovery with variability identification ③ is

Table 4.1 MobileMedia variants' description

Variant	Intent	Type of Change
V1	MobilePhoto core (YOUNG, 2005)	-
V2	Development of exception handling functionality	Inclusion of non-functional concern
V3	Development of features to count the number of photo visualization, to sort according to frequency, and to edit photo's label	Inclusion of optional and mandatory features
V4	Development of the feature to specify and view favorite photos	Inclusion of optional feature
V5	Development of feature to keep multiple copies of photos	Inclusion of optional feature
V6	Development of feature to send a photo via SMS	Inclusion of optional feature
V7	Development of features to store, to play, and to organize music. Transformation of photo management into alternative feature	Changing one mandatory feature in two alternative features
V8	Development of videos management feature	Inclusion of alternative feature

organized in step ① identify the mandatory elements present in all the variants, step ② identify the optional elements present in only part of the variants, step ③ combine the recovered information from the previous two steps, and step ④ include detailed information from packages and methods.

The PLA presentation ④ is organized in step ⑤ which is the application of the metamodel, step ⑥ include fine-grained information, and ⑦ improve the recovered information. SAVaR provides some techniques to support of this steps.

4.1.1 Purpose of the PLA recovery

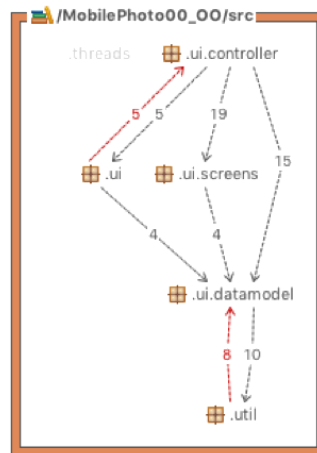
Ducasse and Pollet (2009) grouped SAR purposes into six main goals categories. In the context of this thesis, SAVaR focuses on Redocumenting and Understanding. We selected these topics because they are aligned with our research interest.

The goal of PLA recovery is to reestablish abstractions of the PLA allowing the variability identification at the architectural level of SPL projects. In this way, PLA recovery provides the (re)documentation based on the SPL source code by using bottom-up recovery processes. Based on the recovered information, software engineers could understand the implementation of the variability in a higher level of abstraction which can lead to identifying inconsistencies that are hard to find in the SPL source code.

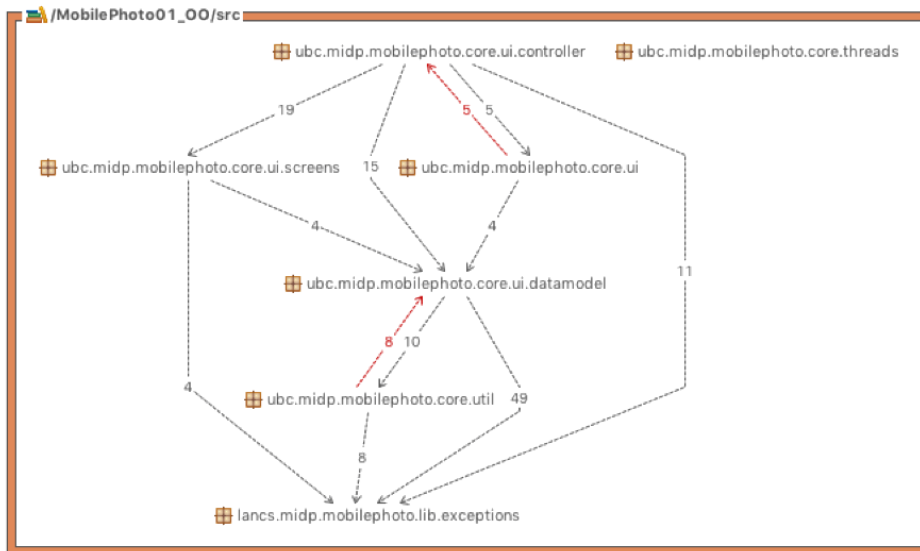
4.1.2 Illustrative Example

Next, we present the MobileMedia (FIGUEIREDO et al., 2008) which is an SPL for applications that manipulate photo, music, and video on mobile devices. The developers used a previous SPL called MobilePhoto (YOUNG, 2005) and added seven change scenarios leading to eight releases.

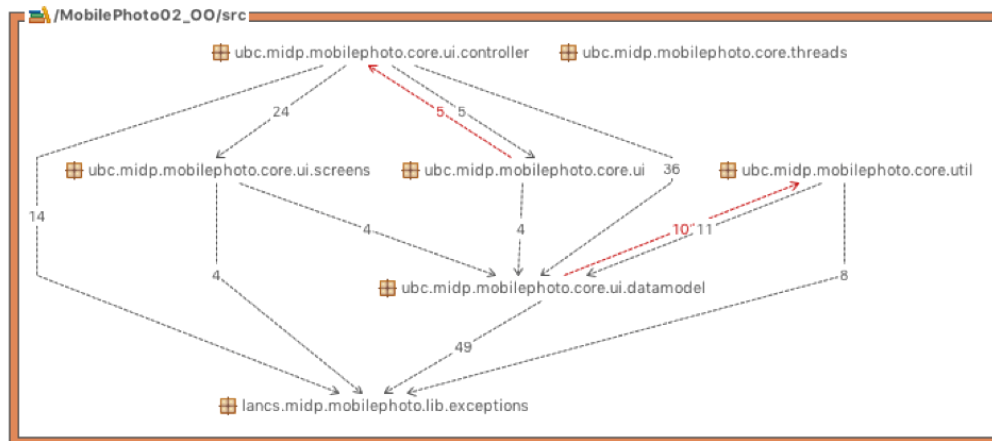
We considered these eight releases as variants and we used them as input to SAVaR. Table 4.1 summarizes the changes made in each release. The scenarios comprise different types of changes involving mandatory, optional, alternative features, and non-functional concerns. The purpose of these variants' changes is to assess the design stability in terms of how implementations of feature boundaries and their dependencies evolved through the SPL variants.



(a) Variant 1



(b) Variant 2



(c) Variant 3

Figure 4.3 Extracted information from three Variants (packages and relations)

4.1.2.1 Variants' source code analysis and extraction

Figure 4.3 presents the extracted information of three variants (V1, V2, and V3) of the MobileMedia project. We used Stan4j tool to extract these dependency graphs. In the case, we are using package representation for describing the development view. The other five variants are available at Appendix B.1.

The dependency graphs nodes denote artifacts and edges denote dependencies. An edge's weight reflects the dependency's strength, which is the number of underlying code dependencies. Moreover, the red lines represent design tangles that is a subgraph with at least two nodes, where each node is reachable from each other.

MobileMedia designs are mainly determined by the use of the Model-View-Controller (MVC) architectural pattern (BUSCHMANN et al., 1996). For instance, we can verify this information by analysing the recovered architecture of the Variants (see Figure 4.3 (a), (b), and (c)), the model is implemented by the package `datamodel`, view is implemented by the package `screens`, and controller by the package with the same name.

4.1.2.2 Application of the PLA recovery

Figure 4.4 shows the development view provided by SAVaR. The PLA encompasses a set of packages. As expected, the modules MVC (`controller`, `screens`, and `datamodel`) are implemented by all the eight variants. For this reason, SAVaR maps them as core elements of the PLA. On the other hand, `exceptions`, `comms`, and `sms` are optional because they are implemented by only some variants.

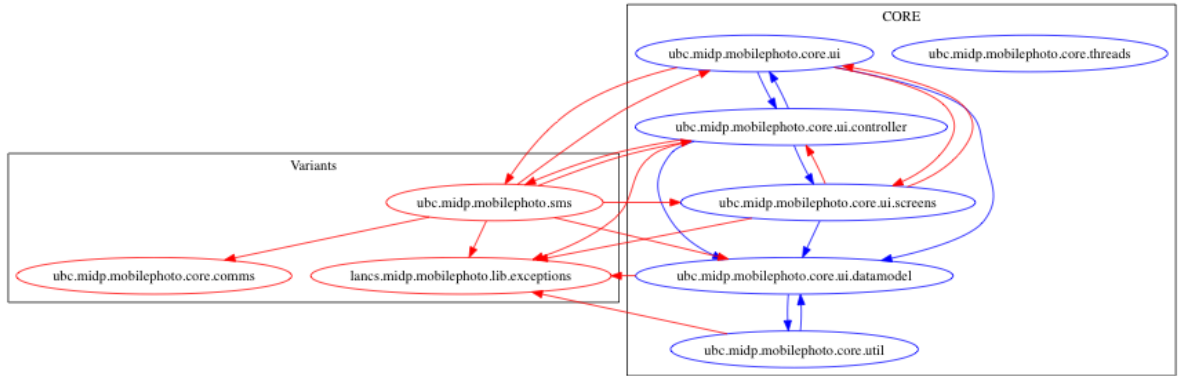


Figure 4.4 Recovered PLA – development view

Figure 4.5 presents the recovered DSM that is a different representation for the development view. Each line and column represent a package. For instance, line 5 and column 5 represent the package `controller`. This package calls classes in the packages `datamodel`, `ui`, and `screens`. The relationship between these packages are mandatory because they are implemented by all the variants. We used the color blue to represent the mandatory packages and relationships. On the other hand, the relationships of `controller` with `exceptions` and `sms` are variable because they are implemented by some variants. We used the color red to indicate the variability. The additional material of the recovered PLA for MobileMedia can be found in Appendix B.1.

	01	02	03	04	05	06	07	08	09
1 <code>lancs.midp.mobilephoto.lib.exceptions</code>									
2 <code>ubc.midp.mobilephoto.core.comms</code>									
3 <code>ubc.midp.mobilephoto.core.threads</code>									
4 <code>ubc.midp.mobilephoto.core.ui</code>									
5 <code>ubc.midp.mobilephoto.core.ui.controller</code>									
6 <code>ubc.midp.mobilephoto.core.ui.datamodel</code>									
7 <code>ubc.midp.mobilephoto.core.ui.screens</code>									
8 <code>ubc.midp.mobilephoto.core.util</code>									
9 <code>ubc.midp.mobilephoto.sms</code>									

Figure 4.5 Recovered DSM of the PLA

4.2 A METAMODEL FOR PLA DESCRIPTION

Figure 4.6 presents the metamodel to support SAVaR. We simplified the metamodel proposed by Thiel and Hein (2002b). We updated the metamodel according to the standard ISO/IEC/IEEE 42010 for describing PLA. Moreover, we included an adaptation of the architecture variability extension and design element extension. We maintained the components used to represent the PLA recovered by SAVaR.

A PLA description (*PLA Description*) is an architecture description enriched with explicit information about the variability model (*Architectural Variability Model*) of the SPL of interest (*SPL-of-interest*). This means that PLA architecture models must be concerned with the representation of architectural variation points (*Architectural Variation Points*). The architectural variability models consist of architectural variability (*Architectural Variability*) that is represented by different architecture views.

In the context of SAVaR, the PLA architectural views are extended with architectural variability representation. The metamodel provides support for the PLA description of SAVaR. The variability information is represented in development views and architectural elements such as packages and classes. We used the variation point specification (*Variation Point Specification*) for describing the optional and mandatory elements. The former (*Optional Element*) represents the elements that are implemented by only some variants and the latter (*Mandatory Element*) represents the ones that are implemented by all the variants.

4.3 PLA RECOVERY

This section introduces SAVaR to PLA recovery, a semi-automatic bottom-up SAR process that comprises two recovery techniques (Section 4.3.1), and a sequence of extract-abstract-present activities (Section 4.3.2). Moreover, we present a set of documented guidelines to help successful PLA recovery with SAVaR in different contexts (Section 4.4).

4.3.1 Techniques

We developed two techniques to address the identification of architectural variability for a given SPL from source code analysis. The first one relies on the extracted architectures of variants while the second technique makes use of `#ifdef` directives in source code.

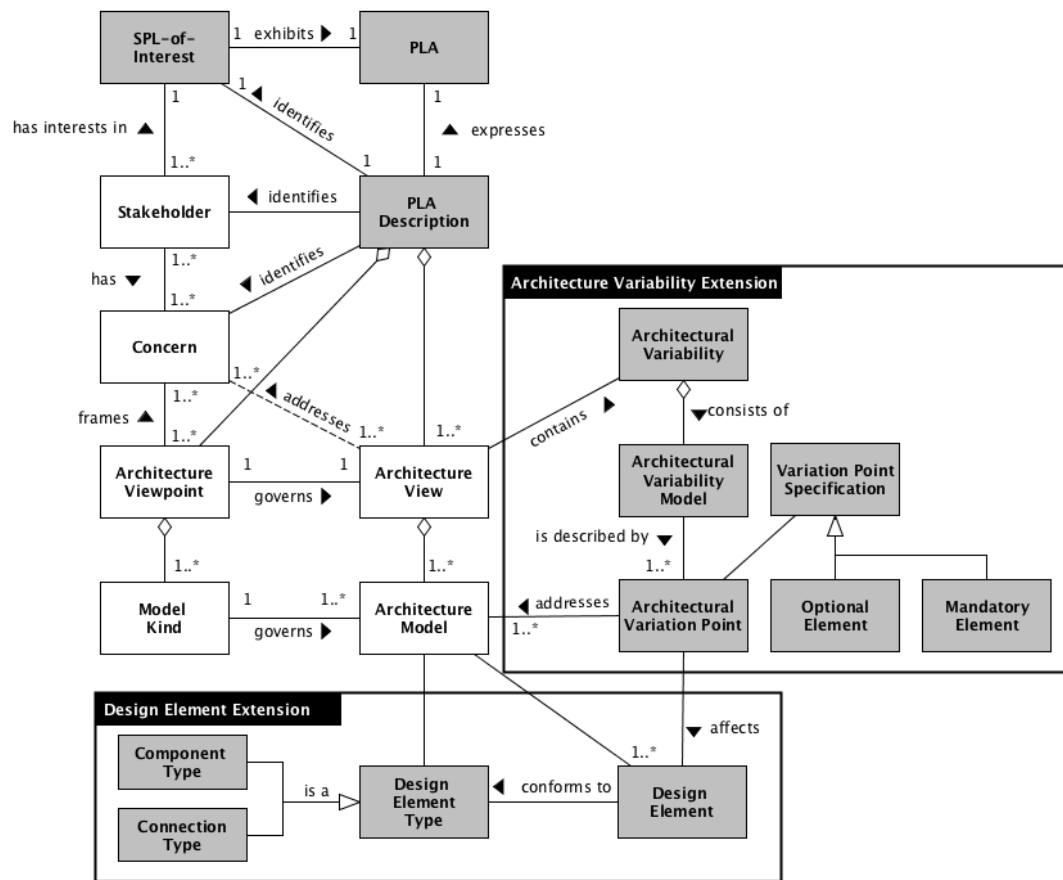


Figure 4.6 Conceptual elements of SPL and SA (adapted from (THIEL; HEIN, 2002b))

Both techniques provide high-level source code models annotated with information about variability. Moreover, we developed a third technique to improve the results of the previous ones. The technique implements a threshold analysis to reduce the variability in the recovered PLA. Next, we provide a discussion of the techniques.

4.3.1.1 Extract-and-Merge: recovery based on merging the extracted architectures of variants.

This technique uses extraction tools to recover a set of architectures, one for each variant. The recovered architecture for each variant instance is internally represented as a set of modules and their relationships. Pattern matching based on module name is used to identify mandatory and optional elements. A module that is present in every recovered variant architecture is labelled as a “mandatory element” while others are labelled as “optional element”. Relationships between mandatory elements are labelled as “mandatory relationship”. The set of mandatory elements and relationships, and optional elements and relationships define the recovered PLA for the SPL and its products (LIMA-NETO et al., 2015). This technique is useful for PLA recovery in the context of a SPL project implemented with different variability mechanisms for products portfolio that preserves the name of mandatory elements in the different products, for legacy systems of the same domain, and so on.

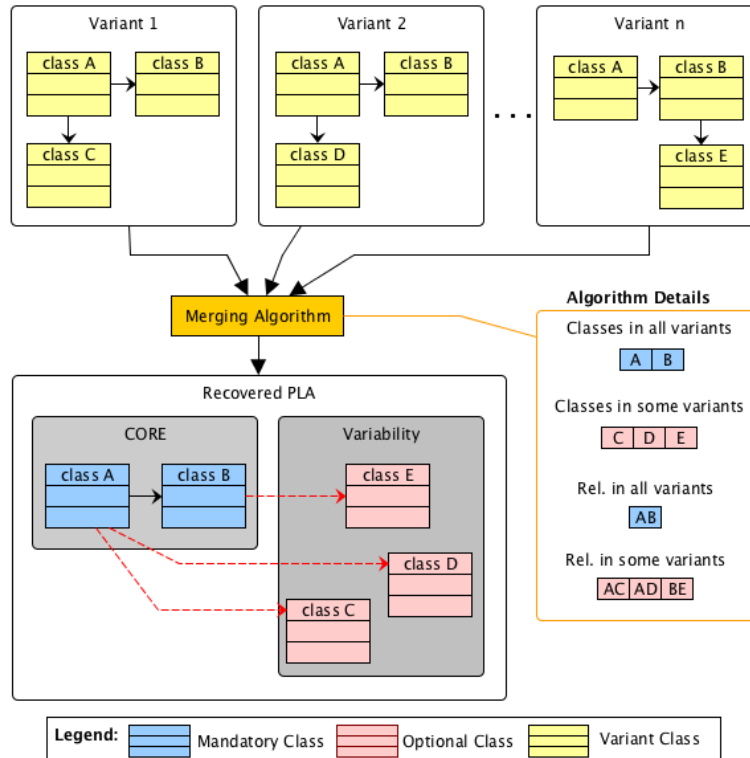


Figure 4.7 PLA recovery based on variants' architecture

Figure 4.7 illustrates this technique. The core of the technique is a merging algo-

rithm (CARDOSO et al., 2017b) whose inputs are the architectures extracted from the variants and the output is the recovered PLA. The technique identifies and groups mandatory elements and their relationships (*e.g.* see classes A and B, and the relationship between them), and organizes optional elements and relationships (*e.g.* classes C, D, and E, and the afferent relationships, represented with a dashed line).

4.3.1.2 #ifdef-Analysis: recovery based on the analysis of #ifdef directives

This technique focuses on PLA recovering in the context of SPL projects that use conditional compilation as a mechanism to implement the variability. The technique identifies if an element or relationship is either mandatory or optional by means of the analysis of the `#ifdef` directives found in the source code.

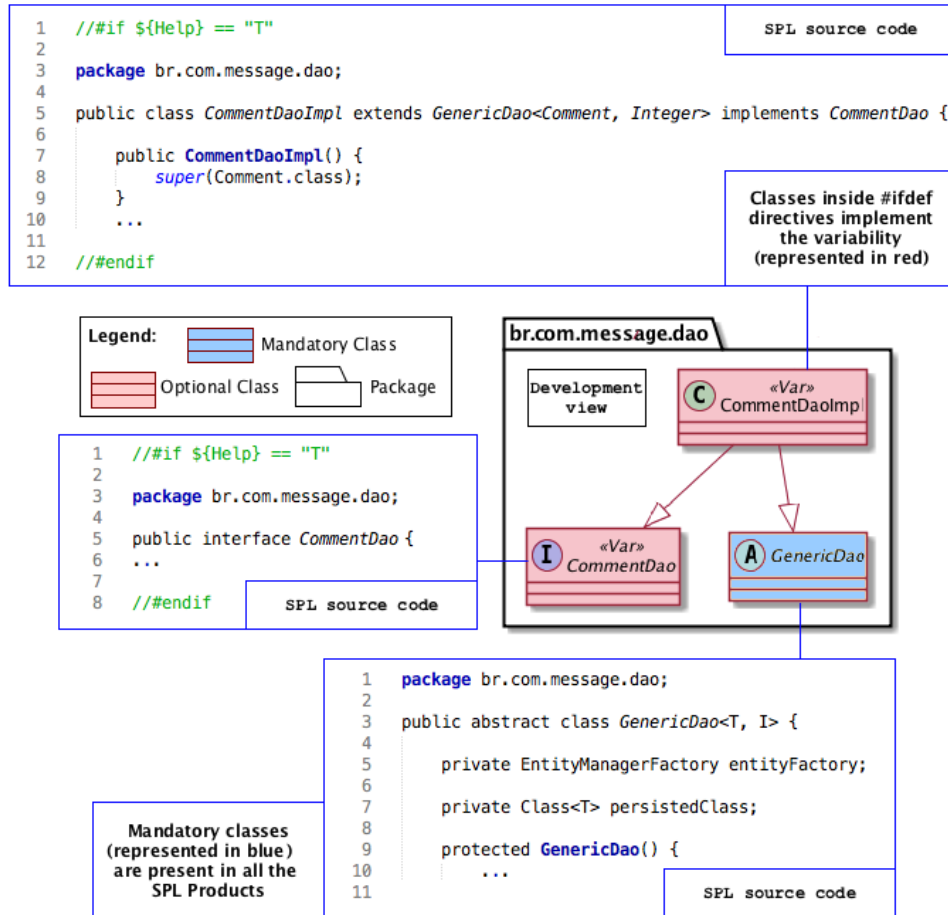


Figure 4.8 PLA recovery based on `#ifdef` directives.

Figure 4.8 presents an excerpt of variability identification based on `#ifdef` directives. Classes `CommentDao` and `CommentDaoImpl` are defined inside `#ifdef` directives and therefore, implement the variability. This means that they are optional classes that may be found in a subset of SPL products (if selected during product configuration). On

the other hand, classes defined outside the `#ifdef` directives (*e.g.* `GenericDao`) are mandatory and will be present in every SPL product.

4.3.1.3 Threshold analysis

We implemented a threshold analysis technique that identifies *exclusive elements*, that is, variable elements that are implemented by only a small number of variants, and whose representation may result in a cluttered visual for the recovered PLA. Therefore, the technique suggests exclusive elements (and relationships) as candidates to be removed from the PLA recovery process. The threshold analysis keeps potential variability explosion under control, by excluding exclusive elements and keeping elements implemented in the majority of the products.

To determine the threshold configurations to be taken into account during the analysis, we rely on the frequency of each element (package, class, or relationship) in the variants. The technique prioritizes the mandatory (core) elements and the elements implemented in most variants.

Figure 4.9 presents one example of the threshold technique. We applied the technique using three threshold values (13%, 38%, and 51%) from MobileMedia project. We selected these values from the report provided by SAVaR to present an example of how the threshold reduce the amount of optional elements. By applying the threshold of 13%, the elements implemented in less than 13% variants are not considered in the outputs. For instance, the package `BaseMessaging` is implemented in less than 13% of the variants. When the threshold value is raised the package is not considered by SAVaR. As a result, the DSM's size was reduced in 19% (see Figure 4.9 (a)). The next technique execution reduced the DSM's size in 50% (see Figure 4.9 (b) – Threshold: 38%). In addition, the third execution of the threshold reduced the DSM's size in 59% (see Figure 4.9 (c) – Threshold: 51%).

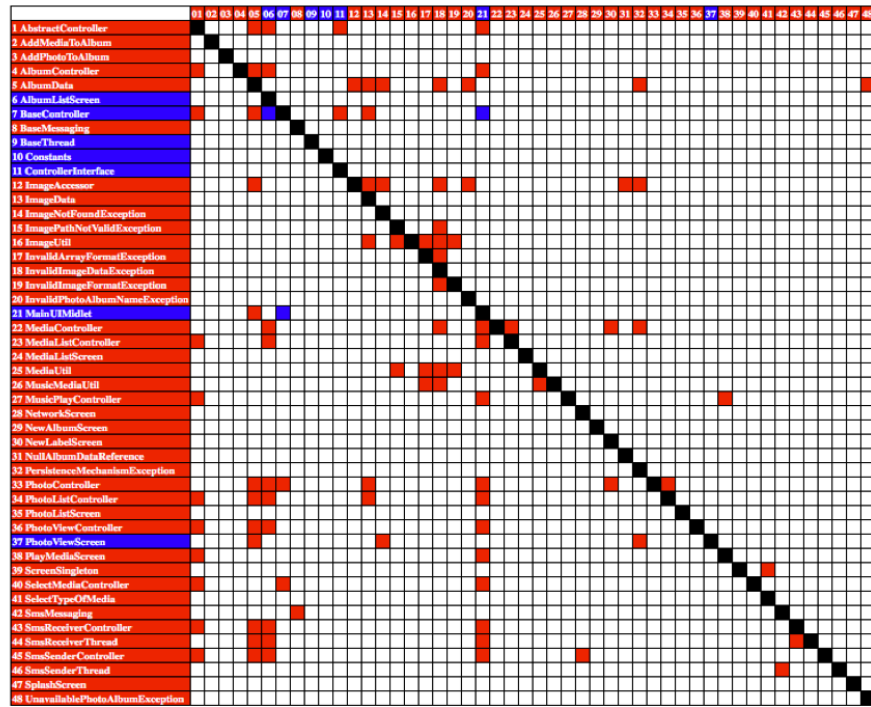
Moreover, the threshold provides (i) a clear view of the recovered PLA for architects, (ii) supports developers' implementation and maintainability tasks by focusing on the mandatory elements and the elements implemented in most of the variants, and (iii) allows recoverers to improve variability identification.

4.3.2 Activities

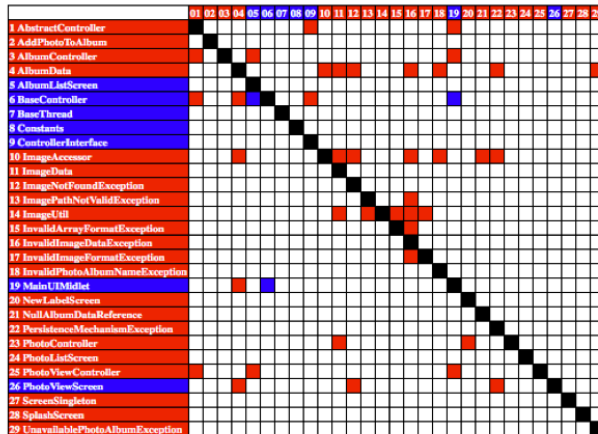
SAVaR supports a lightweight recovery workflow based on a sequence of extract-abstract-present activities (TILLEY; PAUL; SMITH, 1996). The recovered PLA is represented in accordance with a reference metamodel (MOON; CHAE; YEOM, 2006; LIMA; CHAVEZ, 2016) that provides a comprehensive conceptual basis for variability at the architectural level.

4.3.2.1 Extraction

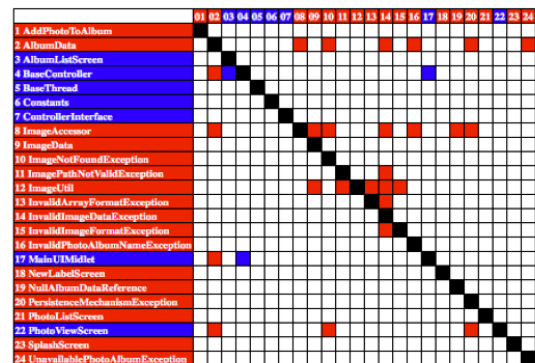
Several analysis and extraction tools that work for single systems can be used to extract



(a) Threshold: 13%



(b) Threshold: 38%



(c) Threshold: 51%

Figure 4.9 Application of the threshold in the MobileMedia (classes)

information from individual variants, for instance, Stan4J² (Java), Struct101³ (Java), Understand⁴ (C and Java), Analizo⁵ (Java, C, C++, and others), and PlantUML Dependency⁶ (Java). These tools provide low-level source code models for products implemented in one or more programming languages. The reuse of extraction tools may require the implementation of adapters to deal with different input/output formats.

The selection of an analysis or extraction tool is based on the programming language and/or the mechanism used to implement the variability. For instance, Stan4J can be used to extract source code models from SPLs implemented in Java, and cppstats⁷, a toolsuite for analyzing cpp-preprocessor-based SPL, can be used to extract information from products with variability implemented by means of `#ifdef` directives.

4.3.2.2 Abstraction with Variability Identification

The two recovery techniques presented in Section 4.3.1 are used to identify (i) mandatory elements and relationships and (ii) optional elements and relationships. After variability identification, the variability-aware architectural models that comprise a PLA can be generated.

In SAVaR, the architectural models conform to the specification of a variability-aware architectural metamodel (THIEL; HEIN, 2002a; MOON; CHAE; YEOM, 2006; LIMA; CHAVEZ, 2016), that includes modeling elements for representing mandatory and optional modules and their relationships. Modules can be packages and classes, and other units of modularization.

According to Galster *et al.* (GALSTER et al., 2013), the metamodels provide a trade-off between the separation of concerns and integration of relevant variability information in single models. It is possible to merge metamodels of views with other metamodels to incorporate the needed variability information. Finally, the metamodels ensure traceability from the SPL source code with the PLA.

We highlight that the implementation of a specific element can differ from one variant to the other. However, this variability happens in low-level (e.g. the same method implemented using different algorithms). Our objective is to identify the variability in architectural level. In this context, the low-level variability (fine-grained granularity) did not affect the structure of the PLA.

Figure 4.10 shows the PLA recovery inputs and outputs organized in three layers. The bottom layer (Layer 0) provides source code as input to allow the extraction of structural information. As outputs of SAVaR, the middle layer (Layer 1) raises the abstraction level using the information of classes, and the top layer (Layer 2) gathers the classes in packages.

Layer 2 provides one consolidated package diagram of the PLA. Then, Layer 1 presents a consolidated class diagram for each package identified in Layer 2. For instance, **Package**

²<<http://stan4j.com/>>

³<<http://structure101.com>>

⁴<<http://scitools.com/>>

⁵<<http://www.analizo.org>>

⁶<<http://plantuml-depend.sourceforge.net>>

⁷<<http://fosd.net/cppstats>>

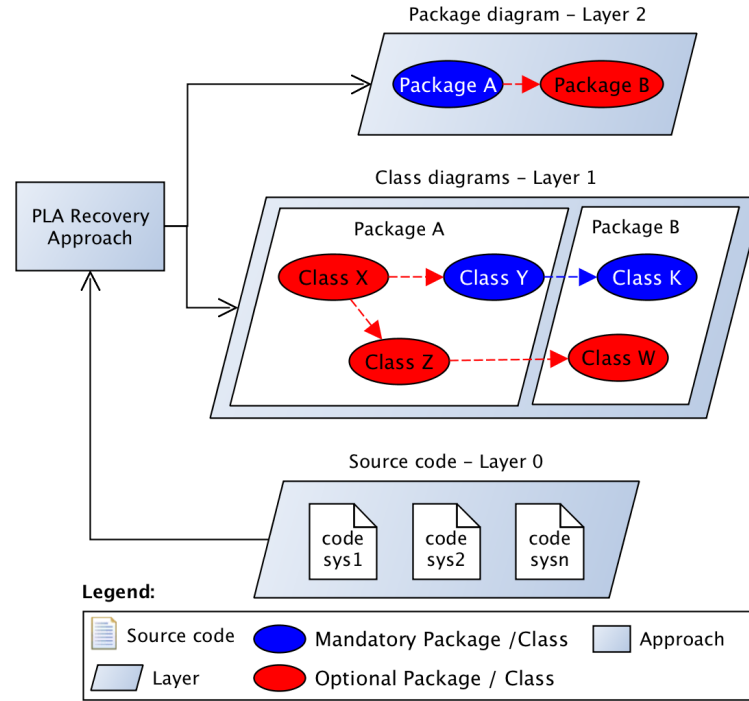


Figure 4.10 PLA Recovery inputs and outputs.

A is a mandatory package because it is implemented in every Variants (Layer 2). Layer 1 allows the developers to check the classes that implemented **Package A** (Classes X, Y, and Z).

4.3.2.3 Presentation

The recovered PLA can be presented using visual notations for development views (*e.g.*, UML class diagrams, module dependency graphs or DSMs) enriched to explicitly provide information about architectural variability and related assets.

Figure 4.8 presents a development view that shows a package (`br.com.message.dao`) with three classes and inheritance relationship. The development view explicitly documents classes as either mandatory or optional, based on the information recovered from the source code annotated with `#ifdef` directives. Moreover, the development view is enriched with boxed code excerpts of the corresponding SPL annotated source code.

4.3.3 Supporting Tools

In our previous study (CARDOSO et al., 2017b), we developed and evaluated the PLAR tool to support SAVaR. In this thesis, we reviewed the PLAR tool implementation and extended the tool by implementing new features for supporting SAVaR. The tool automates the variability identification and serves as a concept proof for the techniques presented in Section 4.3.1. Its inputs are a set of MDG (Module Dependency Graph) (MANCORIDIS et al., 1998) files which represent the elements and relationships from the variants; its

output is the PLA in different formats. These MDG files are extracted from the variants source code through dependency extraction tools such as: Analizo, DependencyFinder, STAN4J, Struct101 and so on.

The recovered PLA provides information about modules (which can be represented by the system classes) (MANCORIDIS et al., 1998) and relationship used in the architectural description of traditional systems, and information about commonalities and variabilities to describe the PLA.

The PLAR tool implements the Algorithm 1. The algorithm receives as input a set of variants. All variants are analyzed, individually, to identify the architectural elements. They are classes, interfaces, abstract classes, relationship, and so on. If the element has not been identified yet, then it will be added to the list of PLA elements; else the technique counts the number of occurrences of the element in the variants. After the analysis, the algorithm verifies whether each element is either present in every variant (core) or not (variable).

Algorithm 1 is organized in two parts. The first part analyzes all the variants and gathers the information. The process is done incrementally, the variants are analyzed in ascending order according to the files' names. However, the order does not influence the recovery results. The second part labels the indentified information from part one and performs the metrics collection.

4.3.3.1 Tool Functionalities PLAR Tool implements the following features:

1. **PLA Recovery Support** - The PLAR tool automates the variability identification. It compares a set of variants' structural information and identifies the core and variable elements;
2. **PLA Quality Evaluation support** - The tool calculates metrics using the recovered PLA information.
3. **PLA Visualization Support** - The PLAR tool provides representations in different formats, which are inputs to visualization tools. The tool provides visualizations that allow the creation of the PLA module view, highlighting the commonality and variability, of DSM, and class diagrams. Each visualization was produced to follow the color the pattern of the module view; red represents variability while blue represents commonality;
4. **Threshold Technique Support** - The tool implements the threshold technique. It filters the information according to a determined value. The threshold separates the PLA elements and provides a clear view for the PLA. The objective is to reduce the amount of noise in the PLA representation. For instance, by defining the threshold value of 5%, the elements implements in less than 5% of the variants are not considered in the SAVaR outputs.

	Input: P , a collection of N variants, $N > 1$.
	Output: PLA, a collection of common and variable architectural elements
1	begin
2	foreach <i>variant P_i in P</i> do
3	foreach <i>element e or relationship r found on P_i</i> do
4	if <i>the element e or the relationship r has been already identified in another variant P_j, $i \neq j$</i> then
5	Add +1 on the number of occurrences of the element e or relationship r ;
6	Update the information about the variant that contains the element e or relationship r ;
7	end
8	if <i>The element e or relationship r has not been identified</i> then
9	Insert the element e or relationship r on the PLA collection;
10	Mark the element e or relationship r as identified;
11	end
12	end
13	end
14	foreach <i>element e or relationship r identified on the PLA</i> do
15	if <i>the element e or relationship r is on all variants</i> then
16	Mark the element e or relationship r as "core";
17	else
18	Mark the element e or relationship r as "variable";
19	end
20	end
21	end

Algorithm 1: PLA recovery algorithm

4.4 GUIDELINES FOR PLA RECOVERY

We documented some guidelines to help practitioners willing to use SAVaR in their working settings. Each guideline describes a realistic PLA recovery scenario, and provides a solution, with steps, hints for improving the PLA recovery and other helpful information. Table 4.2 presents the name and intent of each guideline.

4.4.1 Clone-and-Own

Intent. Recover the PLA from a set of variants.

Problem.

It's not uncommon for small and medium-sized companies to adopt SPL using a *clown-and-own* strategy, by copying, adding or removing functions from existing products (RUBIN; CHECHIK, 2012). This approach leads to *ad hoc* product port-

Table 4.2 Guidelines for PLA Recovery.

Guideline	Intent
Clone-and-Own	Recover the PLA from a set of variants implemented using clone-and-own strategy.
Generate Variants	Recover the PLA from a set of variants automatically generated.
Analyze <code>#ifdefs</code>	Recover the PLA from source code annotated with <code>#ifdef</code> directives.

folios of multiple yet similar variants (FISCHER et al., 2014). With the growth of products portfolio, the management of variability and reuse becomes more complex (SHATNAWI; SERIAI; SAHRAOUI, 2015). A PLA for the SPL could be recovered from its variants and be used to tame complexity, support SPL adoption, 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 you 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;
- Lack of tools to support the variability identification;
- Each cloned variant may have evolved independently from others;
- The possible number of different variants may be very large, up to 2^N , where N is the number of optional and independent features;
- Project development assets evolve, but the architecture documentation do not reflect the improvements over the life cycle.

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.

Use the **Extract-and-Merge** technique with an existing set of variants. Execute the threshold technique to improve the recovered PLA according to development interests.

Pre-conditions.

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. Verify the elements names and eliminate information specific to a variant;
4. Perform the variability identification using the PLAR tool support to automate the process;
5. Analyze the outputs provided by **SAVaR** (collected metrics, report, and visualization of the recovered PLA);
6. [optional] - Run the threshold analysis according to the specified value (integer);
7. [optional] - Go back to step 5 and verify the metrics values.

Post-conditions.

1. Set of reports, metrics, DSMs, development views, UML diagrams from the recovered PLAs according to threshold values.

Hints for improving the PLA recovery.

- Use the report provided by **SAVaR** to manually identify the elements' frequency;
- Execute the threshold technique according to the elements' frequency;
- Select an extraction tool that supports the programming language of the project. Some tools present limitations regarding the output type. For instance, Understand tool provides the recovered information in tabular data while Stan4j uses trivial graph format.

Trade-offs.

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

Suggestion of tool chain.

Figure 4.11 presents the suggested tool chain to support the Clone-and-Own guideline. Download of the variants' source code from the repository using Git. Then, extract the structural information of all the variants using Analizo or Stan4J. We used Sublime tool to eliminate information specific to the project such as the project name and folder organization. We execute the PLAR tool to identify the variability and core elements of the PLA.

Related Guidelines.

Generate Variants, Analyze #ifdefs.

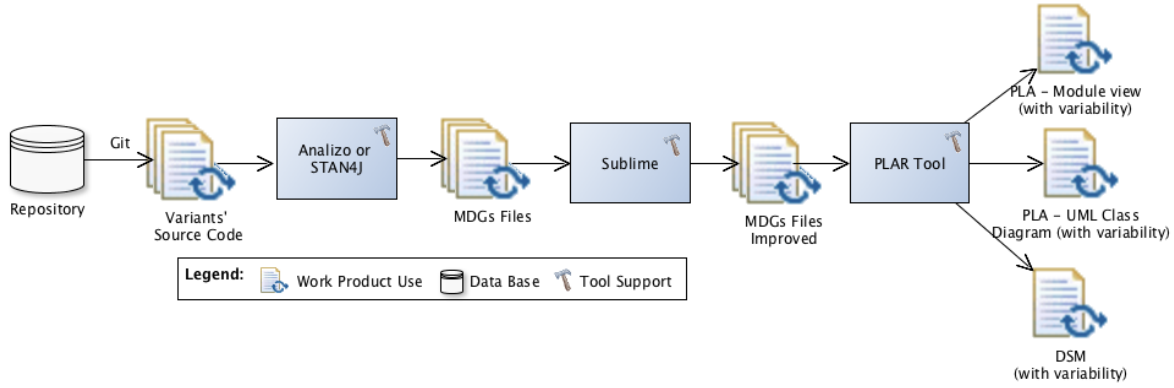


Figure 4.11 Sugestion of tool chain for Clone-and-Own guideline

4.4.2 Generate Variants

Intent. Recover the PLA from a set of variants automatically generated.

Problem.

Although the architecture description is part of the SPL adoption process, not all projects have a PLA documented. The PLA recovery can help developers with the SPL evolution and maintenance tasks. One alternative is to recover the PLA based on the SPL products source code. If variants (in this case SPL products) are generated by tools such as FeatureIDE (MEINICKE et al., 2016), configuration of variants and variant generation are necessary to populate the variants set. However, it could lead to the optional feature problem that describes a common mismatch between variability intended in the domain and dependencies in the implementation (KäSTNER et al., 2009). When this situation occurs, some variants that are valid in the domain cannot be produced due to implementation issues. In this context, we may ask:

How do you recover a software architecture that unveils variability and commonality for a portfolio of generated variants?

This problem is difficult because:

- The possible number of different variants may be very large, up to 2^N , where N is the number of optional and independent features;
- Project development assets evolve, but the architecture documentation do not reflect the improvements over the life cycle.

Yet, solving this problem is feasible because:

- The set of variants can be reduced to using algorithms;
- The existing SAR techniques can be adapted.

Solution.

Generate variants that represent different configurations and use them as input to a technique that combines an existing set of variants to recover the PLA (**Clone-and-Own**). Use the **Extract-and-Merge** technique with an existing set of variants. Execute the threshold technique to improve the recovered PLA according to development interests.

Pre-conditions.

1. SPL source code that allow the automatic generation of the SPL variants.

Steps.

1. Generate the variants using a variant generator tool;
2. Select an extraction tool to recover the variants' structural information;
3. Perform the variability identification using the PLAR tool support to automate the process;
4. Analyze the outputs provided by **SAVaR** (collected metrics, report, and visualization of the recovered PLA);
5. [optional] - Run the threshold analysis according to the specified value (integer);
6. [optional] - Go back to step 4 and verify the metrics values.

Post conditions.

1. Set of reports, metrics, DSMs, development views, UML diagrams from the recovered PLAs according to threshold values.

Hints for improving PLA recovery.

- Use the report provided by **SAVaR** to manually identify the elements' frequency;
- Execute threshold technique to identify possible improvements in the SPL development;
- Select an extraction tool that supports the programming language of the project. Some tool presents limitations regarding the output type.
- When compiling every single product is not feasible, FeatureIDE variant generator provides a T-wise sampling (HENARD et al., 2014) that creates a set of relevant variants, based on the SPL feature model.

Trade Offs.

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

- A variant generator tool is needed;
- Extract all the variants in projects with a high number of optional features could be cumbersome;
- Select a determinate number of relevant variants can impact on the PLA output precision.

Suggestion of tool chain.

Figure 4.12 presents the suggested tool chain to support the Generate Variants guideline. Download of the SPL' source code from the repository using Git. Then, generate all the variants automatically using Feature IDE or using the Ant build. Extract the structural information of all the variants using Analizo or Stan4J. We execute the PLAR tool to identify the variability and core elements of the PLA.

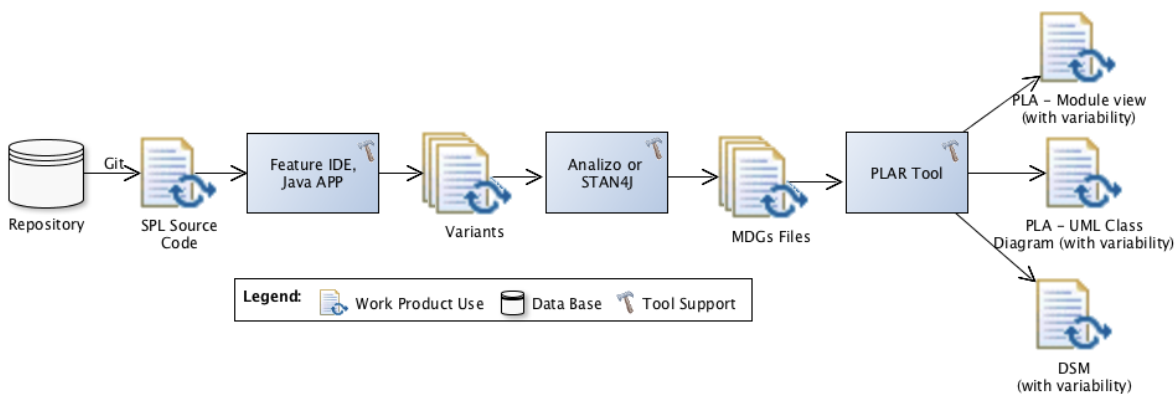


Figure 4.12 Sugestion of tool chain for Generate Variants guideline

Related Guidelines.

Clone-and-Own, Analyze `#ifdefs`.

4.4.3 Analyze `#ifdefs`

Intent. Recover the PLA from source code that contains `#ifdef` directives.

Problem.

Highly configurable software systems allow the efficient and reliable development of similar software variants based on a common code base. The C preprocessor CPP, which uses source code annotations that enable conditional compilation, is a simple yet powerful text-based tool for implementing such systems. However, since annotations interfere with the actual source code, the CPP has often been accused of being a source of errors and increased maintenance effort.

How do you recover a software architecture that unveils variability and commonality for variants created based on `#ifdefs` directives?

This problem is difficult because:

- Projects may be large and complex;
- The source code needs to be inspected or analyzed to identify `#ifdefs` directives;

Yet, solving this problem is feasible because:

- Conditional compilation is a rather used technique to variants' configuration and is language-independent;
- There are several tools that support the analysis of `#ifdef` directives;

Solution.

Use the **#ifdef-Analysis** technique to recover the PLA for a SPL that uses `#ifdef` directives to implement variability and guide product generation. Execute the threshold technique to improve the recovered PLA according to development interests.

Pre-conditions.

1. Project source code developed using `#ifdef` directives available.

Steps

1. Select an extraction tool to recover the variants' structural information;
2. Select a tool to extract `#ifdefs` information;
3. Perform the variability identification using the PLAR tool support to automate the process;
4. Analyze the outputs provided by SAVaR (collected metrics, report, and visualization of the recovered PLA);
5. [optional] - Run the threshold analysis according to the specified value (Feature name or File name);

Post conditions.

1. Set of metrics, DSMs, and development views from the recovered PLAs.

Hints for improving the PLA recovery.

- Apply the filters according to feature name or file name.
- Create a Docker⁸ container and use servers in the cloud to reduce the time and effort to extract the structural information.

Trade Offs.

⁸<<http://www.docker.com>>

- Lack of tools to support large-scale projects such as Linux kernel. Tools used in this context should be robust. They commonly presented some issue such as run out of memory and stopped to work;
- The PLA recovery demanded time and processing effort. The extraction process can take days to finish.

Suggestion of tool chain.

Figure 4.13 presents the suggested tool chain to support the Analyze `#ifdefs` guideline. Download of the SPL' source code from the repository using Git. Then, extract the structural information of the project using Analizo. Moreover, extract `#ifdefs` information using `cppstats` tool. We execute the PLAR tool to identify the variability and core elements of the PLA.

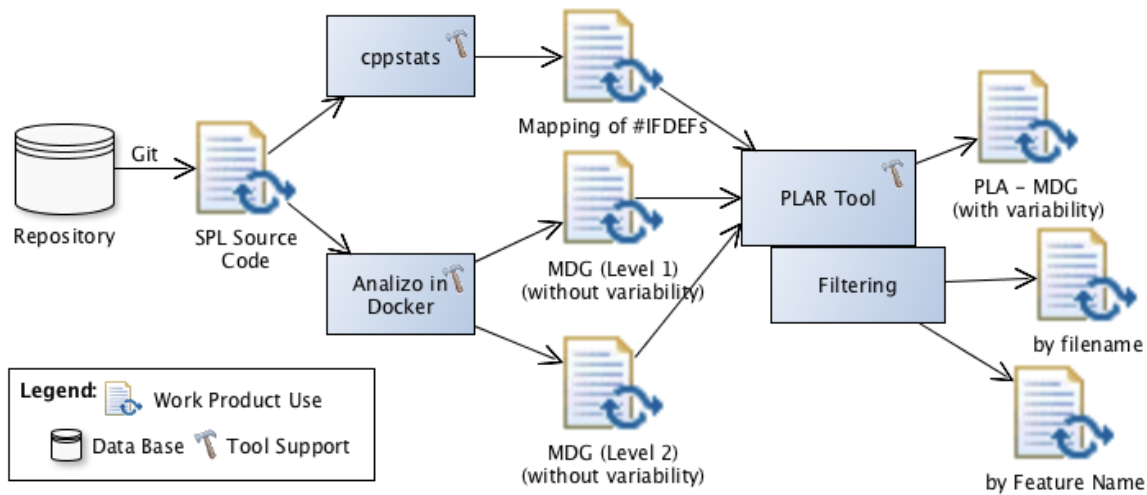


Figure 4.13 Sugestion of tool chain for Analyze `#ifdefs` guideline

Related Guidelines.

Clone-and-Own, Generate Products.

4.5 RELATED WORK

Chapter 3 presented a SMS that investigated the relationship between PLA and SAR. Such study provided evidence that the majority of studies addressed some aspects of SPL. However, they lack detailed information to support the PLA recovery. Existing proposals to PLA recovery were found (SHATNAWI; SERIAI; SAHRAOUI, 2015, 2016; LINSBAUER; LOPEZ-HERREJON; EGYED, 2016), and research trends and gaps on the subject were identified.

Shatnawi, Seriai and Sahraoui (2015) proposed an approach to recover the PLA based on the comparison of components recovered from different versions of the same SPL. The authors relied on the Formal Concept Analysis (FCA) to analyze the variability and created a variability model.

Later, they extended the study in (SHATNAWI; SERIAI; SAHRAOUI, 2016). They focused on the recovery of components (combining classes and interfaces). In our thesis, different versions were not analyzed. We extracted the structural information from products of the SPL, and from the SPL source code directly. Moreover, we also collected information about the variability in classes, packages, and their relationship.

Linsbauer, Lopez-Herrejon and Egyed (2016) presented an approach for extracting information from sets of related product variants. The authors extracted structural information from the SPL products source code. Moreover, they compared the information to recover the Feature Model (FM). We implemented a similar recovery process. However, instead of recovering the FM, we compare the recovered architecture from SPL products to recover the PLA.

Existing approaches do not provide support – lack of guidelines (see Chapter 3) – for different PLA recovery scenarios (commonly they focus only in one context of the PLA recovery). Moreover, we provide **SAVaR** to allow the integration of different variability management approaches in heterogeneous environments.

4.6 CHAPTER SUMMARY

This Chapter introduced **SAVaR** to support the PLA recovery. In this way, we proposed a series of phases and activities detailing each one of the steps of **SAVaR**. Moreover, we provided a set of guidelines to help SPL architects, developers, and recoverers to perform these activities as a means to systematize the PLA recovery and ensure a correct traceability between the SPL source code and its architecture.

Further, we discussed the variability identification in details. To the best of our knowledge, these two tasks are critical for defining the PLA and consequently for PLA recovery. Few studies addressed the variability at the architectural level of SPL projects. Finally, **SAVaR** gathers information of each PLA recovery execution to facilitate future recoveries, allow the evolution of the PLA recovery process, and provide the PLA recovery replication. In the next Chapter, we present an exploratory study performed to evaluate and improve **SAVaR**.

The only true wisdom is in knowing you know nothing – Socrates

EXPLORATORY STUDY ON PLA RECOVERY

This Chapter presents an empirical evaluation aiming to analyze the activities performed during PLA recovery in the context of SPL projects. This Chapter describes an empirical study conducted in an academic environment with students from a software reuse graduate course. We carried out the study by following the guidelines for conducting experiments in software engineering defined by Wohlin et al. (2012).

The remainder of this Chapter is organized as follows: Section 5.1 presents the exploratory study design. Section 5.2 presents the execution. Section 5.3 discuss the analysis and interpretation. Section 5.4 discusses our findings and threats to validity. Section 5.5 presents the related work. In Section 5.6, we present the conclusions.

5.1 DESIGN

We investigated the PLA recovery in two scenarios. The first scenario focused on the PLA recovery directly from the SPL source code. The subjects developed the PLA using `#ifdefs` directives. The second scenario focused on the PLA recovery based on the combination of the recovered architecture of SPL variants (products).

We aimed to investigate the application of **SAVaR** and the adaptation of existing SAR recovery tools and techniques in the context of SPL projects. For this purpose, we reported the study by following the guidelines for conducting experiments in software engineering (WOHLIN et al., 2012).

5.1.1 Planning

We applied the GQM method (SOLINGEN et al., 2002) to define quantitative measures and provide the objective assessment.

5.1.1.1 Goal

The objective of this study is to analyze how to recover PLAs by applying the SAVaR for the purpose of understanding with respect to its effectiveness and reliability from the point of view of developers and recoverers in the context of SPL projects in an academic environment.

5.1.1.2 Questions

To achieve this goal, we defined the following Research Questions (RQ):

- **RQ 1: Does SAVaR provide a precise and reliable version of the implemented PLA?**

In Chapter 3 the majority of studies addressing PLA recovery focused their empirical evaluation on the application of case studies. Therefore, these studies did not evaluate the recovered information from the perspective of both architects and developers. Inspired by the work of Garcia et al. (2013), in which authors verified the recovered information with architects or developers, we asked the SPL developers the following two sub-research questions:

- *RQ 1.1: Does SAVaR recover the PLA correctly?*

By answering this question, we verified if SAVaR provided a precisely recovered information. We asked the subjects if the recovered PLA was correct and if they found inconsistencies regarding the project source code and the recovered PLA. Such a feedback could enable us to improve the SAVaR's precision in recovering the PLA.

- *RQ 1.2: Does SAVaR provide a reliable PLA?*

To answer this question, the recoverers presented the recovered PLA for the developers' verification. Then, we asked the subjects if they trust the information recovered by SAVaR. This feedback is relevant because the developers understand their SPL project source code and know the variability implementation. If something went wrong during SAVaR application, it could compromise SAVaR reliability.

- **RQ 2: How much details are needed to represent the recovered PLA?**

According to Galster et al. (2013), choosing a variability representation and the amount of variability is a complex task. By answering this question, we aim to understand the issues related to identifying the amount of variability in the recovered PLA. In this way, this RQ was split into the following sub-questions:

- *RQ 2.1: Do the developers prefer more detailed information regarding the recovered PLA?*

To answer this question, we asked the developers if the amount of information provided by SAVaR is enough. We verified if they needed more information related to the variability at the architecture level. In this way, we can calibrate the proposal to provide more or less information according to the subjects demands.

- *RQ 2.2: Does the amount of information provided by SAVaR compromise the PLA general view?*

On the other hand, in this question, we verified if the amount of information provided by SAVaR compromise the general view of the PLA. By answering this question, we checked if SAVaR led to information overload. In other words, the subjects may have problems to understand the PLA overall view.

- **RQ 3: Do the metamodels (for PLA design) support on the understanding of the recovered PLA?**

In a preliminary investigation (LIMA; CHAVEZ, 2016), we identified that metamodels have been proposed and used for defining languages and notations for representing and managing the variations of a PLA. In the context of our exploratory study, we investigated how these metamodels support understanding of the recovered PLA. This question has been split into two more specific research questions:

- *RQ 3.1: Do the metamodels helped on the recovered PLA understanding?*

By answering this question, we verified if the metamodels represent the recovered PLA also allow the understanding of the recovered information. First, we asked the subjects to select a metamodel from the list identified in our previous work (LIMA; CHAVEZ, 2016). Then, we asked them if the chosen metamodel helped in the understanding of the recovered PLA.

- *RQ 3.2: Do the metamodels would support the SPL project development?*

One of the main objectives of SAVaR is to connect the SPL source code with the PLA and allow the variability traceability in the architectural level. The metamodel can be used as a mechanism (*i.e.* bridge) of connection between the recovered PLA and the SPL source code. For this reason, we asked the subject if the metamodels would provide support for their development tasks.

5.1.2 Study Design

According to the steps and guidelines (Generate Variants and Analyze `ifdefs` defined in Chapter 4, we organized the study design using SAVaR phases: (i) Information collection, (ii) Information Extraction, (iii) PLA Recovery, and (iv) Presentation of the Recovered PLA. Figure 5.1 presents the study design phases and the subjects involved in each phase. Developers and recoverers performed the activities of phase one and four, the recoverers were responsible to perform phases two and three.

To perform phase 1, we analyzed the SPL source code and other assets (*e.g.* Feature Model) produced by the subjects, and we also interviewed them using data collection method based on interviews (LETHBRIDGE; SIM; SINGER, 2005). At the end of this step, we identified the need to gather more information about the variability implementation such as the type of binding time implemented, the tools and techniques they used to allow the variability and products instantiation, understand how they configure the products.

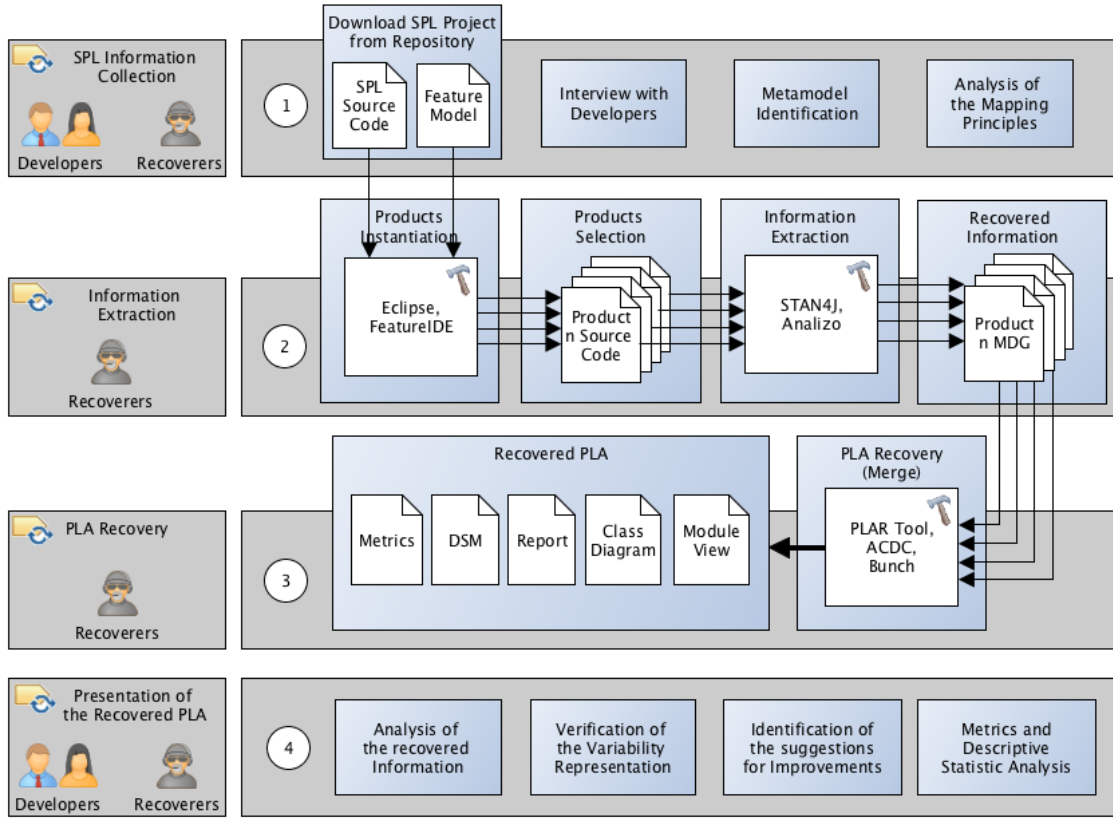


Figure 5.1 Study Design Phases

We considered this information relevant because such design decisions might affect the SPL project development, maintenance, and evolution. Consequently, these decisions would impact on the PLA definition, variability representation, and may influence the variability identification.

In Phase 2, the recoverers performed the products' instantiation by using FeatureIDE and the structural information extraction from the SPL source code by running the extraction tool. In this way, the recovery technique used different types of input. For this reason, we developed a set of input converters to allow the synchronization of extraction tools with recovery techniques. Phase 3 compares the extracted information from the products and provide the outputs regarding the recovered PLA – including metrics, reports, diagrams, and so on.

In Phase 4, the developers verified the recovered information. The recoverers asked them if the recovered PLA and variability representation were correct. We also collected suggestion of improvements regarding the provided information and SAVaR.

5.1.3 Exploratory Study materials

We designed and used the following materials: Consent Form, Background, Feedback Questionnaires, list of metamodels to support PLA design, a set of information from the

recovered PLAs – UML class diagram, Module view, DSM, Report, and metrics.

5.1.4 Subjects

We applied convenience sampling (WOHLIN et al., 2012) in this study. The subjects were graduate students (one Ph.D. Student and twelve M.Sc. Students) from Advanced Topics in Software Engineering course (*i.e.* Software Reuse Fundamentals: Theory and Practice) at Federal University of Bahia, Brazil. The Ph.D. Student and one master student acted as recoverers and the other eleven students were organized in two groups.

The classes hosted the ‘experimental lab’ including the selection of subjects and performing the activities of the exploratory study. This course was designed to explain the fundamental concepts and principles of Software Reuse focusing on the development of SPL projects.

Moreover, the students acted as SPL developers being responsible for developing an SPL project from scratch. They provided information necessary to perform the PLA recovery. Finally, they also verified the recovered information to allow improvements of the recovered PLAs.

The students defined two groups to implement the SPL projects. The division was based on the subjects’ expertise in software development and their experience in industry. The motivation for the subjects to attend the study was based on the assumption that they would have the opportunity to use the recovered PLA as an asset to improve their SPL projects.

Two students acted as recoverers. According to Garcia, Ivkovic and Medvidovic (2013) the recoverer is an engineer producing (*i.e.* recovering) the architecture – the PLA in the context of our study. They analyzed the existing SPL information necessary to perform the PLA recovering activities.

5.1.5 The Study Projects

The subjects developed the SPL projects used in this study. Table 5.1 presents the projects and metrics – related with implementation such as interfaces, methods, packages, and classes. The first one consisted of a SPL for the e-commerce domain called SPL Web Store¹. The second one focused on the instant message domain called SPL Message². More details about the recovered information can be found in the projects website.

5.2 EXECUTION

The subjects carried out the activities involved in the exploratory study. Table 5.2 presents the study execution agenda. It describes the number of days to perform each activity, the hours dedicated to the task, the subject responsible, and the phase.

¹<<http://homes.dcc.ufba.br/~crescencio/WebStoreSPL/>>

²<<http://homes.dcc.ufba.br/~crescencio/SPLMessage/>>

Table 5.1 SPL Projects – Metrics

SPL Project	Domain	TLOC	#D	#R	#P	#F	#I	#M	#PAC	#C
SPL Web Store	e-commerce	5.5k	10	2	10	27	0	422	6	87
SPL Message	instant mes.	3.2K	1	2	10	19	12	243	8	46

Legend: [TLOC] Total Lines of Code, [#D] Number of Developers, [#R] Number of Recoverers, [#P] Number of Products used to recover the PLA, [#F] Number of features, [#I] Number of Interfaces, [#M] Number of Methods, [#PAC] Number of Packages, [#C] Number of classes

Table 5.2 Study Execution Agenda

Days	Activities	Length	Responsible	Phase
1 day	Interview with Developers	4 hours	Dev. and Rec.	1
1 day	Metamodel Identification	4 hours per subject	Developers	1
1 day	Information Extraction	4 hours per project	Recoverers	2
2 days	Recovery of the PLAs	1 day per project	Recoverers	3
1 day	Presentation and Verification of the Recovered PLA	4 hours per group	Dev. and Rec.	4
1 day	Improvements in the Recovered PLAs	1 hour per project	Recoverers	4

5.2.1 Procedure

Initially, the subjects became aware of the study. In the first day, we interviewed the developers to identify how they implemented the variability in the SPL project. We also discussed the products instantiation and the techniques they used to allow the products configuration.

In the second day, we provided a list of metamodels to support the PLA design defined in our previous work (LIMA; CHAVEZ, 2016). We asked the subjects to select at least one metamodel to support the PLA design understanding of their project. The subjects selected the metamodels considering their SPL context and domain.

In the third and fourth days, the recoverers used the information raised in the previous days to carry the PLA recovery. In the fifth day, the recoverers presented the recovered information for the developers. The developers verified the recovered PLA – using the metamodel as template – and suggested improvements for the recovery. They also filled a feedback questionnaire about the recovered PLA. In the last day, the recoverers applied the suggestions of improvement and provided the recovered PLA for the developers.

5.3 ANALYSIS AND INTERPRETATION

In this section, we present the analysis and interpretation of the results. We verified how the subjects answered the research questions. We used the Likert scale (ALBAUM, 1997) to collect the subject answers (*i.e.* Strongly disagree, Disagree, Neutral, Agree, and Strongly agree).

5.3.1 RQ 1: Does SAVaR provide a precise and reliable version of the implemented PLA?

Figure 5.2 presents the subjects' answers for RQ 1.1 and RQ 1.2. In the former, we asked the subjects if SAVaR recovered the PLA correctly. As can be seen, they agreed that the recovered PLAs are in conformance with the project source code. In the later, we asked the subjects if they trust on the information provided by SAVaR.

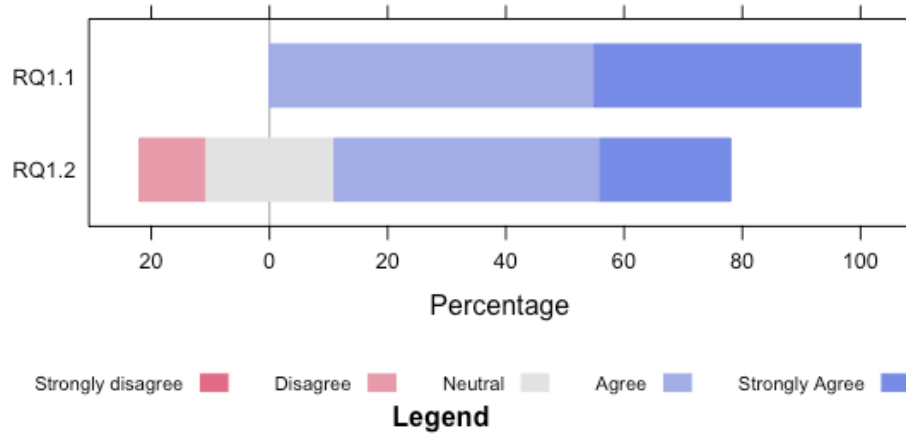


Figure 5.2 Research Question 1 Answers

5.3.2 RQ 2: How much detail is needed to represent the recovered PLA?

Figure 5.3 shows the answer of the subjects for RQ 2.1 and RQ 2.2. In RQ 2.1, we asked the subjects if they preferred more detailed information in the representation of the recovered PLA – such as packages, abstract and concrete classes, interfaces, and so on. We presented two recovered PLAs UML class diagrams, the first one organized the recovered PLA in packages (*e.g.* dao, facade, and so on), the second ignored these details. Then, we asked what representation they preferred, and the majority of the subjects preferred the detailed information against a more abstract representation.

Moreover, in RQ 2.2, we asked the subjects if the amount of the recovered information provided by SAVaR compromise the PLA general view. As can be seen in Figure 5.3, 24% of the subjects affirmed that the recovered information compromise their understanding of the PLA broader view.

5.3.3 RQ 3: Do the metamodels (for PLA design) support on the understanding of the recovered PLA?

In Figure 5.4, we present the subjects' answers regarding the RQ 3.1 and 3.2. In the first RQ, we asked the subjects to choose a metamodel from the list defined in our previous study (LIMA; CHAVEZ, 2016). Then, we verified if the selected metamodel helped them to understand the recovered PLA. The majority of the subjects (64%) agreed that the metamodel supported their understanding of the recovered information.

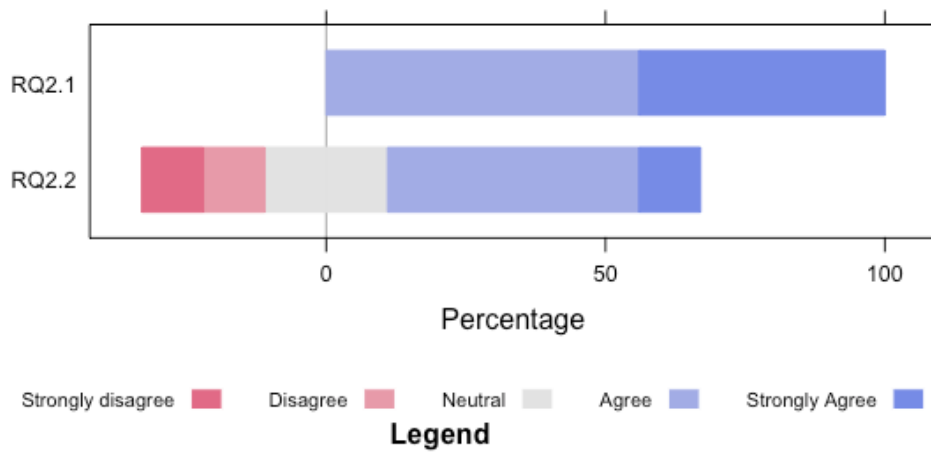


Figure 5.3 Research Question 2 Answers

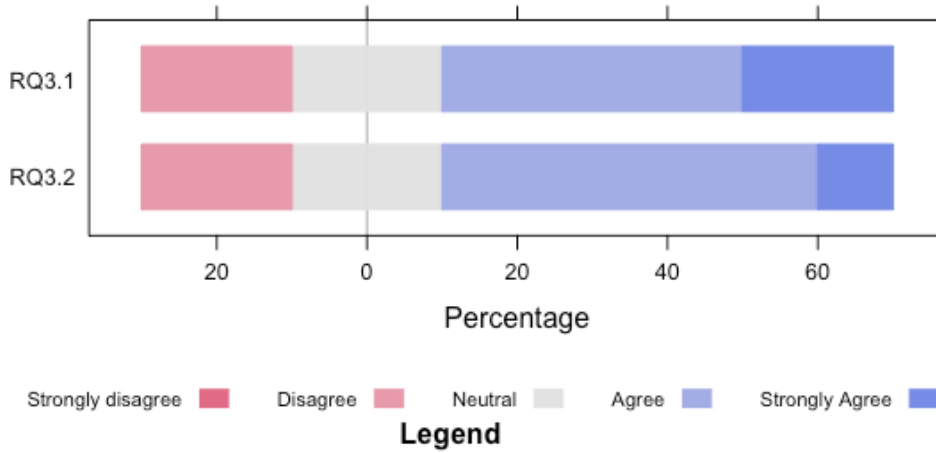


Figure 5.4 Research Question 3 Answers

Regarding the RQ 3.2, the subjects answered if the metamodel would support their development activities. Similarly to RQ 3.1, 56% of the subjects agreed that the metamodel would provide support to their development activities.

5.3.4 Feedback

Moreover, we asked some questions about their personal experience understanding the recovered PLA. They are complementary to the previous research questions and gave feedback on aspects that would be improved upon, by comparing the experience with possible opportunities to improve the results they reported by using SAVaR.

Figure 5.5 shows the results from feedback questions 1 and 2. In the first feedback question, we asked the subjects if they needed help to understand the information of the recovered PLA. Half of the subjects needed help, and the other half understood the recovered information without asking for support. In the second feedback question, we

asked them if the views provided by SAVaR helped in the maintainability of the project source code.

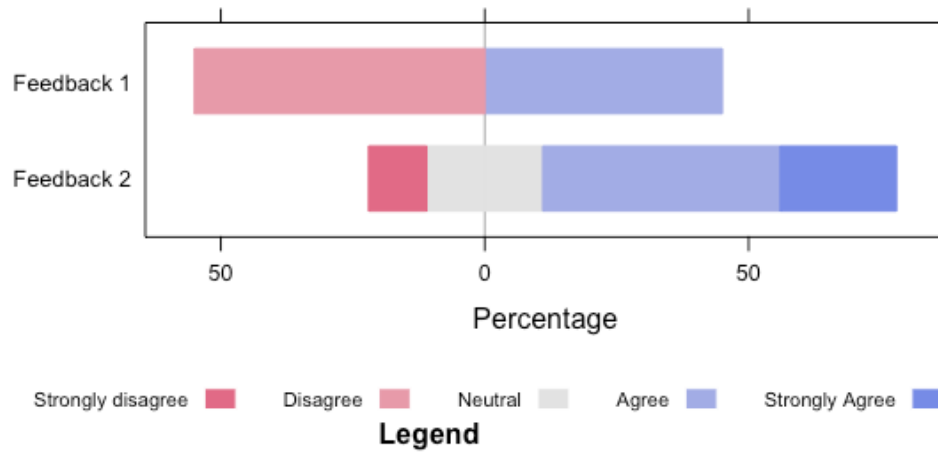


Figure 5.5 Approach feedback part 1

Figure 5.6 shows the results from feedback questions 3 and 4. In the third feedback question, we asked the subjects if they would use SAVaR in their professional daily activities. As can be seen in Figure 5.6, the majority of the subjects (65%) would use SAVaR in their day-to-day tasks. In the fourth feedback question, we asked the subjects if they would suggest the use of SAVaR to their colleagues. Once again, the majority of the subjects (72%) would recommend SAVaR for their colleagues.

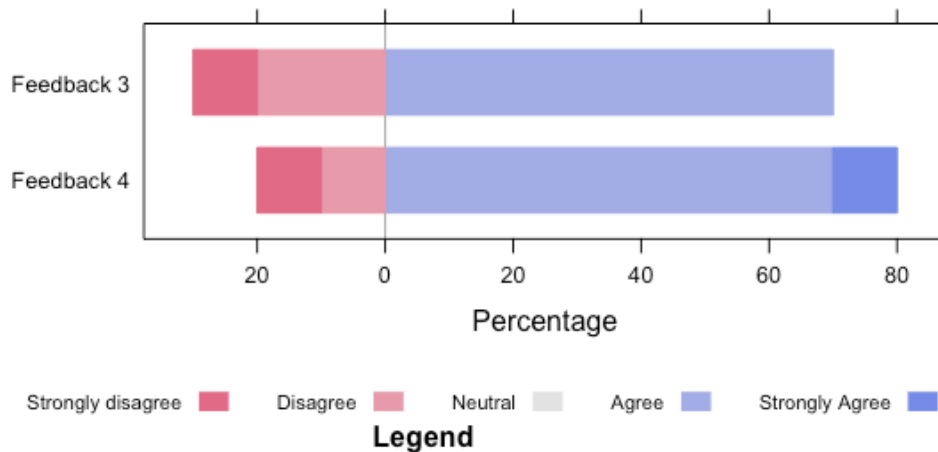


Figure 5.6 Approach feedback part 2

5.3.5 Metrics Analysis - Descriptive Statistics

To provide extra information of the recovered PLAs, the subjects analyzed the collected metric from their projects. Table 5.3 presents the metrics values for the recovered PLAs.

The **SSC** values of the projects were similar (SPL Web Store: 0.70 and SPL Message: 0.63). The same happened with **SVC** values (SPL Web Store: 0.30 and SPL Message: 0.36). The result indicates 30% of the projects' elements implements the variability. Moreover, the SPL Web Store has 20 optional classes, 57 optional relationships, 48 mandatory classes, and 122 mandatory relationships. On the other hand, SPL Message has 22 optional classes, 40 optional relationships, 38 mandatory classes, and 54 mandatory relationships.

Table 5.3 Recovered Metrics from the PLAs

SPL Project	SSC	SVC	RSC	RVC	CO	OR	CM	MR
SPL Web Store	0.70	0.30	0.68	0.32	20	57	48	122
SPL Message	0.63	0.36	0.57	0.42	22	40	38	54

Legend: **[SSC]** Structure Similarity Coefficient, **[SVC]** Structure Variability Coefficient, **[RSC]** Relation Similarity Coefficient, **[RVC]** Relation Variability Coefficient, **[CO]** ClassOptional, **[OR]** OptionalRelation, **[CM]** ClassMandatory, **[MR]** MandatoryRelation

5.3.5.1 Descriptive Statistic Analysis

Another collected metric was the Component Reuse Rate (**CRR**) and Relation Reuse Rate (**RRR**). We did not register these metric in Table 5.3 because it provides a measure for each PLA element and relation. To make it clear to visualize the **CRR** and **RRR** values, we created the boxplot regarding the Component Reuse Rate per group in Figure 5.7 and the boxplot regarding the Relation Reuse Rate per group in Figure 5.8. Group 1 gathers the information from SPL Web Store project and Group 2 gathers the information from SPL Message project.

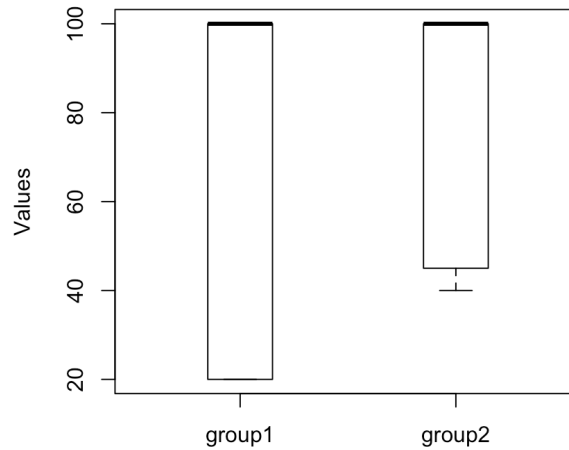


Figure 5.7 Boxplot – Component Reuse Rate per group

The **CRR** value was similar in both SPL projects (see Figure 5.7). On the other hand, we identified a variation regarding the **CRR** presented in the boxplot (see Figure 5.8).

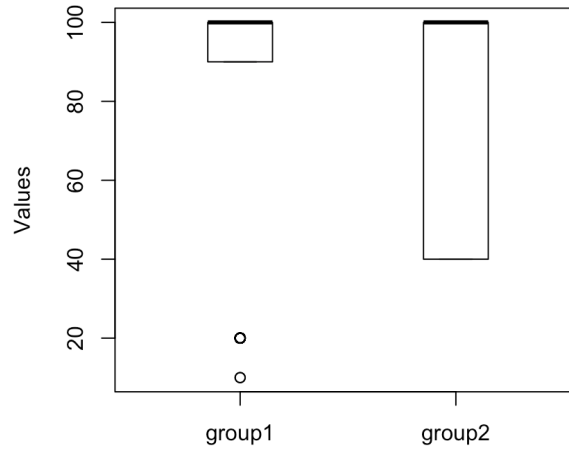


Figure 5.8 Boxplot – Relation Reuse Rate per group

Moreover, we performed statistical tests to verify the relationship among the values. The tests were primarily presented for a significance level of 5%.

We applied the Wilcoxon/Kruskal-Wallis test (NATRELLA, 2010) to compare the CRR and RRR values from group 1 and 2. Regarding the CRR, the *p-value* ($p = 0.8$) was higher than the significance level. In other words, there was no difference among the groups CRR values. The same happened with the RRR values ($p = 0.5$).

5.4 DISCUSSION

In the following, we provide a summary of the main findings, limitations to the review, and threats to validity.

5.4.1 Main Findings

During our study, we confirmed the difficulties to find approaches (including tools and techniques) available to support the PLA recovery in the context of SPL projects. When they were available, we did not identify guidelines (*i.e.* documentation) to help stakeholders in the application of the recovery activity.

For this reason, we created **SAVaR** and developed the guidelines to support the PLA recovery. We also implemented some tools and guidelines to support **SAVaR**. By applying **SAVaR** in the context of SPL projects, we identified improvements opportunities and gaps in **SAVaR** that were raised by the subjects.

The experience from the subjects varied from just one year working in industry until subjects between fifteen and thirty years of experience. In both cases, they had the first contact in the development of SPL project. Regarding their experience, the beginning of the development was harder than in a traditional project. However, at the end of the projects, they agreed that SPL could be used in their professional activities because of the benefits SPL engineering could provide them with.

Regarding the execution of the study, they considered that if the metamodel to design PLA and **SAVaR** were used from the beginning of the project, their understanding and

development of SPL project could be more efficient and effective. The metamodel would provide the support to define the PLA, and SAVaR would allow the synchronization of the developed SPL source code with the PLA and the variability traceability.

Although the effectiveness provided by the SAR tools and techniques, we identified that the communication with the developers during the application of SAVaR still necessary because it helped in the identification of some project decisions that were not registered. For instance, during the interview, the developers explained for the recoverers how they configured the products and implemented the variability. They also provided the variants configurations.

5.4.2 Variability Identification

During the exploratory study, we observed that the variability identification in architectural level is not considered by recovery tools and techniques from single systems. For this reason, we focused our efforts on understanding this phenomenon during the application of SAVaR in the context of SPL projects. We considered this task in SAVaR and verified how it worked in our study.

The variability identification varies according to the variability mechanism used to implement the SPL project. For this reason, we proposed the variability identification by analyzing the `#ifdef` directives. In the second technique, we proposed the merging algorithm to identify the variability among the SPL products. During the comparison of the extracted information from the products, the algorithm identified elements and relationships that were present in the architecture of the products.

We categorized the elements present in all products as the “core” elements of the PLA and elements found in a subset of the products as “variable” elements. In some cases, core elements presented were involved in variable relationships. Variability was also implemented inside methods. This information was not explicitly represented since it did not provide any architectural relevant information.

5.4.3 Amount of details in the PLA recovery

Whenever we implemented the traceability of architectural variability, we had the concern to avoid adding unnecessary information in the documentation. Choosing a variability representation and the amount of variability in the PLA is not trivial. According to Galster et al. (2013), “too much variability can cause project failure”. In general, variability in software should satisfy needs inside a domain or market, and add as little extra complexity as possible (GALSTER et al., 2013).

We observed this concern during the exploratory study execution. We perceived that some subjects complained when we asked them to verify the PLA “big picture” (using the module view).

The subjects used the different views (class diagram, module view, and DSMs) provided by SAVaR to identify the ideal amount of recovered information (and variability identification). They analyzed the core elements, the variable ones, and their relationship.

Most of the subjects preferred the DSM because it provided a simpler view of the recovered PLA. Figure C.1 and Figure C.3 show the recovered DSMs from the SPL

projects used by the subjects in the exploratory study. DSMs provide a broader view of the PLA and facilitate the visual identification of some patterns.

Moreover, SAVaR recovered information about packages and classes. We also created a representation with variables and methods. However, the visualization almost tripled in size and we identified that low-level information did not affected the PLA structure. In other words, such information did not impact in the visualization of the PLA (see Figure 5.10).

5.4.4 Filtering the recovered PLA

Due to the subjects' complaints regarding the amount of information provided by SAVaR, we extracted a simplified view of the PLA. We focused on selecting an optional feature (e.g. Comment) and its relationships. As Figure 5.9 shows, we identified the Model-View-Controller (MVC) structure represented by the model (M), facade (V), and dao (C) packages. Moreover, the classes from the features package activate the variability implementation "inside" the MVC context.

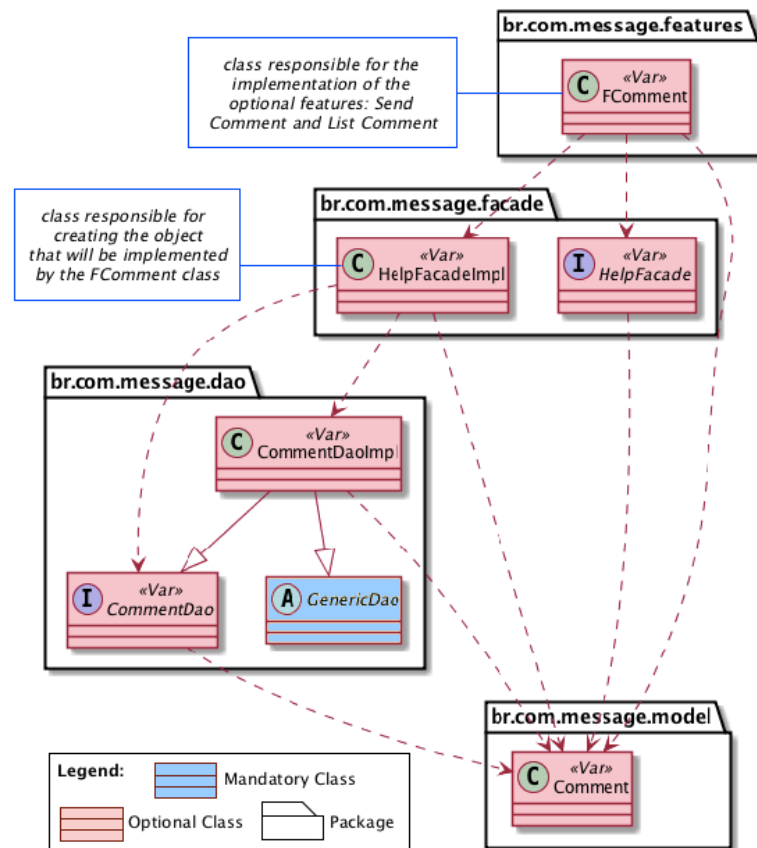


Figure 5.9 Recovered Information isolated by the feature Comment

We recovered information such as abstract classes – represented with the letter **A** (e.g. GenericDao), interfaces – represented with the letter **I** (e.g. CommentDao,

HelpFacade), and concrete classes – represented with the letter **C** (e.g. FComment, CommentDaoImpl, and HelpFacadeImpl). Regarding the relationship representation, we identified inheritance with a filled line, and functions call with a dotted line.

To exemplify how the amount of details impacted in the PLA representation, we selected one class (i.e. FComment) and we recovered information about methods, variables, #ifdefs directives, and so on from that specific class. As can be see in Figure 5.10, most of the retrieved information are irrelevant for the architectural representation. For instance, information about user interfaces (e.g. JFrame, JButton, JLabel, and so on), constructors, getters, and setters did not impact in the PLA and variability identification.

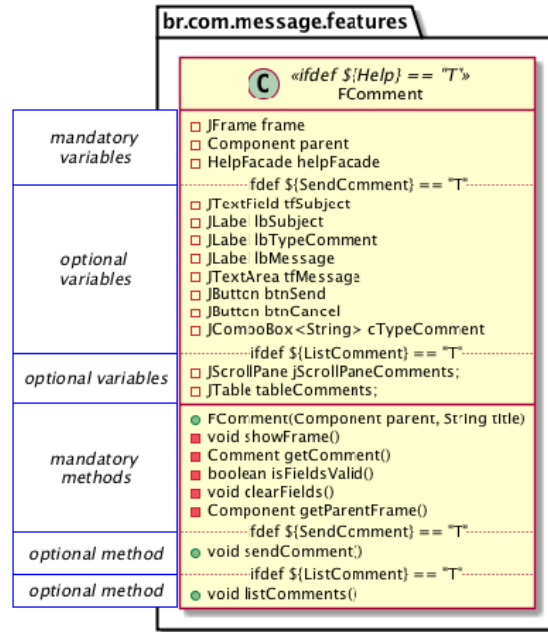


Figure 5.10 Example of class with all the recovered information

On the other hand, we identified that two methods (i.e., listComments() implemented to list the users' comments and sendComments() implemented to send the users' comments) are relevant in the architectural context because they implemented optional features and should be considered in the recovered PLA representation.

5.4.5 Analyzing the recovered PLA based on Metamodels

We asked the subjects to choose a metamodel from the list defined in our previous study (LIMA; CHAVEZ, 2016) to support the definition of their recovered PLA. Due to their lack of experience in PLA design, the metamodels helped on the understanding of the recovered PLA. Combined with their experience in SPL development, the information provided by SAVaR helped them in the identification of architectural flaws in the PLA.

Further, we asked them why they choose that specific metamodel. Most subjects preferred metamodels that represented the PLA according to the feature model. We also identified that they preferred metamodels that provided a high level of abstraction.

The subjects affirmed that the metamodels help them to maintain their PLA. The metamodels extended their previous knowledge to understand the PLA and the SPL development. Moreover, it facilitated the variability management in architectural level.

5.4.6 Analyzing the changes from SAR to PLA recovery

During the adaptation of the existing SAR from single system to SPL, we identified differences between them. For instance, we verified that the developers used the MVC (Model-View-Controller) structure to implement their SPLs.

We observed that the MVC structure remained the same in both single system and SPL. However, the developers adapted the MVC structure by adding a package called *Feature* for activating the optional features.

In other words, when a specific feature is selected, the classes and constructors responsible for implementing that feature appears in the packages from the MVC structure. For instance, Figure 5.9 presents the instantiation of the *Comment Feature*. When this feature was selected in the product configuration, the classes *FComment*, *FCommentDao*, and *CommentView* appears in the respective packages.

Regarding the PLA, some adaptation is needed to migrate an architecture from single system to SPL and the goal of **SAVaR** – as evaluated in this exploratory study – is to identify the variability at architectural level and provide guidelines to support the PLA recovery.

5.4.7 Threats to Validity

Following, we describe some threats to the validity of our exploratory study along with the mitigation strategy for each threat. Below, we list the threats to the validity identified:

5.4.7.1 Conclusion validity

Random heterogeneity of subjects. Wohlin et al. (2012) argued about the risks when the subjects group is very heterogeneous. In the case of our exploratory study, we believed that the subjects heterogeneity regarding the experience in industry allowed the collection of insightful feedback because of the different perspectives. Moreover, the inexperienced subjects demonstrated resistance in performing the study. For this reason, we performed the analysis according to subjects experience in industry.

5.4.7.2 Internal validity

Instrumentation. This is the effect caused by the artifacts used for the exploratory study. *The subjects developed different SPL projects.* In this way, the recovered PLAs were particular to the SPL project. **SAVaR** provides the recovery based on the development, including mistakes introduced by the developers. For this reason, the output provided by **SAVaR** varied according to the project.

5.4.7.3 Construct validity

Evaluation apprehension. Some subjects were afraid of being evaluated because the

study took place during the reuse course. We identified that one subject was not interested in participate on the study. To mitigate this issue, we informed that the exploratory study purpose was to evaluate SAVaR. We also provided this information in the consent form.

5.4.7.4 External validity

Interaction of setting and treatment. This is the effect of not having a material representative of – for instance – industrial practice. On the other hand, some subjects had years of experience in the development of industrial projects and provided their feedback based on the knowledge acquired from these projects.

5.5 RELATED WORK

Garcia et al. (2012) proposed a framework for recovering architectures and then verified the framework application by recovering the architecture of open source projects (GARCIA; IVKOVIC; MEDVIDOVIC, 2013). The studies did not consider SPL projects. Based on these studies, we identified and adapted some steps to perform the PLA recovery. We also proposed new ones that are unique to SPL context.

Shatnawi, Seriai and Sahraoui (2016) performed an exploratory study to evaluate their approach in two SPL projects (*e.g.* Health Watcher and Mobile Media). The authors did not consider the involvement of subjects in the recovery process.

Linsbauer, Lopez-Herrejon and Egyed (2016) performed case studies by applying their approach in the products of 5 SPL projects. However, instead of recovering PLAs, they recovered Feature Models based on the structure of SPL products.

According to the evidence we raised in Chapter 3, we identified only one study providing guidelines to support SAR. However, the guideline did not provide details for PLA recovery. In this way, we used the guidelines defined in Chapter 4 to support the PLA recovery in the context of SPL projects. We worked with SPL developers and recoverers to apply SAVaR. During the recovery, we identified improvements according to the subjects' feedback.

5.6 CHAPTER SUMMARY

In this Chapter, we presented an exploratory study regarding the initial evaluation of SAVaR. Our main intention was to verify points of improvement to evolve SAVaR based in the context of SPL projects working with subjects such as developers and recoverers.

The majority of the subjects agreed that the recovered PLAs provided by SAVaR were in conformance with their SPL project source code. They also trusted in the recovered information. Regarding the amount of information, they preferred more detailed representation. Moreover, only the inexperienced subjects complained that the amount of detail in the recovered information compromised their understanding of PLA general view. Finally, the subjects majority agreed that the metamodel allowed the comprehension of the recovered PLA and supported them on their development tasks.

One of the exploratory study goals was to identified points of improvement based on the subjects feedback. During the application of the first version of SAVaR, we collected

data to perform analysis and allow the replication. Based on the gathered evidence, we improved SAVaR.

For instance, the most cited issue focused on the module view representation of the recovered PLA. The subjects complained that it was difficult to deal with the visualization because of the high number of module and relationships in the representation. In this way, we proposed to break the PLA in smaller pieces and organize the visualization per features.

In the next chapter, we present some studies applying SAVaR after the improvements gathered during this chapter. We recovered the PLAs from 15 SPL projects and verified the quality of the recovered architectures.

We are what we repeatedly do. Excellence, then, is not an act, but a habit – Aristotle

RECOVERING THE PLA OF 15 OPEN SOURCE SPL PROJECTS

This Chapter presents a family of case studies designed to investigate whether PLAs recovered with **SAVaR** can be used to support two important tasks: variability identification and reuse assessment. In each study we recovered the PLA of an open source SPL project with the support of **SAVaR**. The PLAR Tool provided automatic support for recovering the PLA from the SPL products' source code, and calculating the reuse metrics (CARDOSO et al., 2017b).

The remainder of the Chapter is organized as follows. Section 6.1 describes the study design. Section 6.2 presents the study execution, Section 6.3 discusses the results, and Section 6.4 interprets it. Finally, Section 6.5 concludes this chapter.

6.1 STUDY DESIGN

This Section presents the study design shared by 15 case studies, its research questions, hypotheses, and metrics, and discusses the analysis procedure.

We used the Goal/Question/Metric (GQM) approach (SOLINGEN et al., 2002) to define the objective of this family of empirical studies:

Analyze a product line architecture to understand how variability affects architectural recovery with the PLAR tool from the viewpoint of researchers performing the variability identification in the context of SPL projects.

6.1.1 Research Questions

We defined three research questions and associated them to the set of instrumental reuse metrics used (ZHANG et al., 2008; OLIVEIRA-JUNIOR; GIMENES; MALDONADO, 2008).

RQ 1 Does the number of SPL products used in PLA recovery impact the identification of variability in the PLA?

The product generation tool provides the number of SPL products used in PLA recovery. The PLTV metric is used to estimate the total number of variable elements expected in the PLA, and the SVC metric is used to calculate the overall variability of PLA elements.

RQ 2 Is there a relation between the number of optional features in the SPL and the reuse rate of the recovered PLA?

The product generation tool provides the number of optional features in the SPL. We used this information together with the SSC metric to estimate the impact of optional features on the PLA reuse rate.

6.1.2 Hypotheses

In order to answer the research questions, we postulated the following hypotheses:

RQ1 Hypotheses

- H_{0a} – The number of SPL products analyzed does not influence the variability identification.
- H_{1a} – The variability identification is influenced by the number of SPL products analyzed.

RQ2 Hypotheses

- H_{0b} – There is no relation between the number of optional features in the SPL and the number of variation points in the recovered PLA.
- H_{1b} – There is a relation between the number of optional features in the SPL and the number of variation points in the recovered PLA.

6.1.3 Metrics

The quality of a PLA is evaluated based on the metrics presented in Chapter 2. SAVaR collects the following data from the SPL project: number of SPL optional features, number of SPL variation points, number of SPL products (M) and its recovered PLA (number of common components (C_C), number of variable components (C_V), number of common relations (R_C) and number of variable relations (R_V). Data to be collected by PLA assessment includes reuse measurement values for PLA elements and relations.

PLAR generates a dataset containing all the necessary data to proceed with PLA quality analysis according to its reuse rate.

6.1.4 Analysis Procedure

Our method for evaluating the effectiveness of the recovered PLA to support variability identification was based on the correlation analysis of the metrics regarding the number of products and the metrics related to the variability identification (**SVC**, **RVC**, **CO** (**CO**), **OR** (**OR**), and optional features). The analysis examines if the number of products (**M**) influences the variability identification rate of individual PLAs.

To test our hypothesis related to RQ1, we analyzed the correlation analysis results. We applied the non-parametric Spearman rank correlation (DANIEL, 1990), to measure the degree of association between two variables.

Our method for evaluating the effectiveness of the recovered PLA to support reusability evaluation was based on the analysis of **CRR**, which examines whether the number of optional features (**OF**) influences the component reuse rate of PLA elements.

To test our hypothesis related to RQ2, we compared the **CRR** values from 15 SPL projects. We applied the ANOVA (NATRELLA, 2010) to identify if at least one SPL presented different **CRR** value, and the Turkey test (NATRELLA, 2010) to perform a pairwise comparison between the values to answer the research questions.

Finally, our method for evaluating the SPL reuse rate was based on a general analysis of the recovered PLA, and on the individual analysis of each PLA element.

6.2 STUDY OPERATION

6.2.1 SPL Projects analyzed

The SPL projects were selected based on the following criteria: lack of documented PLA and source code written in Java (a constraint imposed by STAN4J).

The selected SPL projects are:

- *Draw Product Line (DPL)* - An SPL for drawing applications;
- *Video on Demand (VOD)* - An SPL for video-on-demand streaming applications;
- *Zip Me* - An SPL for file compression software;
- *Game of Life (GOL)* - An SPL that simulates the board game - game of life;
- *Graph Product Line (GPL)* - An SPL for implementing graph manipulation libraries.
- *Prop4J* - An SPL for arbitrary propositional formulas;
- *BankAccount* - An SPL to manage bank accounts;
- *BankAccountV2* - An improvement of BankAccountSPL with more features;
- *DesktopSearcher* - An SPL that implements programs for indexing and content based searching in files;
- *Elevator* - An SPL used for evaluation in model checking context;

- *ExamDB* - An SPL that implements database management systems that manages exams;
- *PayCard* - An SPL that implements smartcard-payment software products that supports optionally transaction logging and statistics;
- *PokerSPL* - An SPL that implements the poker game variations;
- *UnionFind* - An SPL that implements the UnionFind algorithms.

Table 6.1 summarizes our sample and presents the number of features (mandatory and optional), classes and products of each SPL, and the tool used for product generation.

6.2.2 Preparation

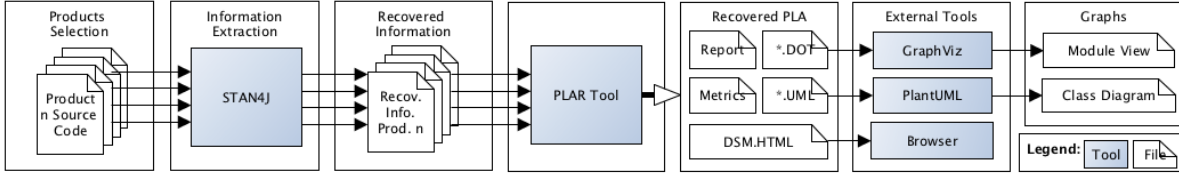


Figure 6.1 The overall recovery process: activities, inputs and outputs.

The selected SPL projects were subject to product generation, MDG extraction with STAN4J, and PLA recovery with the PLAR. Figure 6.1 shows the main activities, inputs, and outputs of the PLA recovery process used in this study. First, we selected the SPL products source code. Second, we performed the Information Extraction – from the SPL product source code – with Stan4J tool. Third, we used the Recovered Information as input for the PLAR Tool. The tool provided the following outputs regarding the Recovered PLA: report, metrics, module view, class diagram, and DSM. Finally, we used some tools such as Graphviz and PlantUML to provide the visualization of the recovered information.

A product configuration is said to be valid when it obeys the configuration model dependencies (APEL et al., 2013). For any SPL with a potential high number of products (*e.g.* Prop4J can have 5K products), we used the T-Wise method (HENARD et al., 2014) to generate only a subset of SPL products. The T-wise method takes as input a configuration model that defines the valid configuration space for the SPL. This model typically includes a set of configuration options, each of which takes a value from a small number of discrete settings, and a set of system-wide constraints among configuration options. Given the model, these methods compute a *t-way* covering array - a set of configurations, in which each valid combination of option settings for every combination of *t* options appears at least once. Finally, some SPL projects (DPL, VOD, Zip Me, and GOL) had existing generated products available, so that we could skip product generation.

6.2.3 Data collection

For each SPL project studied, the PLA was recovered and metrics were collected using the PLAR Tool.

6.3 DATA ANALYSIS

This section presents the statistical analysis of the treatment variables relating to the data items gathered in the study. First, we present some descriptive statistics for the dependent and independent variables; next, we present the analysis of each SPL data – because the SPL projects used different techniques to implement the variability.

In order to evaluate the quality of each recovered PLA, we measured **SSC**, **SVC**, and **CRR** values. The SPL projects with a high **SSC** value and a low **SVC** value indicate that the PLA is mostly composed of common components. Conversely, projects with high **SVC** value and low **SSC** value indicate that the PLA is mostly composed of variable components.

6.3.1 Descriptive Statistics

Table 6.1 SPL Projects analyzed and Metrics collected for PLAs

SPL	#F	#FM	#OF	#P	#C	Ge.	SSC	SVC	RSC	RVC	CO	OR	CM	MR
DPL	5	3	2	12	4	NA	0.5	0.5	0.3	0.7	2	2	2	1
VOD	11	6	5	32	42	NA	0.8	0.2	0.7	0.3	10	23	32	55
Zip Me	7	2	5	32	31	NA	0.8	0.2	0.7	0.3	6	14	25	32
GOL	21	12	9	65	21	NA	0.6	0.4	0.7	0.3	8	11	13	24
GPL	38	18	20	155	15	CD	0.6	0.4	0.4	0.6	6	23	9	16
Prop4J	13	0	13	31	14	FH	0.1	0.9	0.0	1.0	13	50	1	0
BankAccount	6	0	6	24	2	FH	1.0	0.0	1.0	0.0	0	0	2	1
BankAccountv2	8	0	8	72	3	FH	0.7	0.3	0.5	0.5	1	1	2	1
DesktopSearcher	22	6	16	462	41	AH	0.3	0.7	0.1	0.9	30	134	11	14
Elevator	6	0	6	20	5	FH	1.0	0.0	1.0	0.0	0	0	11	29
E-mail	6	0	6	40	3	FH	1.0	0.0	1.0	0.0	0	0	3	4
ExamDB	3	0	3	8	4	FH	1.0	0.0	1.0	0.0	0	0	4	5
PayCard	3	0	3	6	7	FH	0.7	0.3	0.4	0.6	2	5	5	3
PokerSPL	11	2	9	28	8	FH	0.5	0.5	0.3	0.7	4	5	4	2
UnionFind	10	2	8	6	4	FH	1.0	0.0	1.0	0.0	0	0	4	4

Legend: [#F] Features [#FM] Mandatory Features [#OF] Optional Features [#P] Product [#C] Classes [Ge.] Product Generator [NA] Not Available [CD] CIDE [FH] FeatureHouse [AH] AHEAD [CO] ClassOptional [OR] OptionalRelation [CM] ClassMandatory [MR] MandatoryRelation

Table 6.1 presents the metric results of the recovered PLAs. The **SSC** metric is used to calculate the overall similarity of PLA elements; the maximum value is 1. The greater the value of **SSC** the better the reuse rate of the PLA elements (ZHANG et al., 2008). The **SVC** metric calculates the general variability of the PLA elements; the maximum value is 1. The greater the value of **SVC** the worse the reuse rate of the PLA (ZHANG et al., 2008). The **RSC** and **RVC** metrics are similar to **SSC** and **SVC**, respectively. However, they are used to measure the similarity and variability of relations among the PLA elements.

The metrics **ClassMandatory** and **ClassOptional** calculate the number of mandatory and optional classes, respectively. The number of mandatory classes should be greater than the number of optional classes to indicate a better reuse of PLA com-

ponents (OLIVEIRA-JUNIOR; GIMENES; MALDONADO, 2008). Besides, the metrics **OptionalRelation** and **MandatoryRelation** use the same principle to calculate the number of mandatory and optional relations. It is possible to perform these calculations because PLAR analyzes the classes and relations captured in the MDG files.

PLTotalVariability is a metric that estimates the PLA variability. Given that the MDG files were not able to capture detailed information about class methods and attributes, these metrics could not be used at all. Table 6.1 shows the sum of **ClassOptional** and **OptionalRelation** metrics.

The **CRR** metric is missing from the overview presented in Table 6.1. This metric provides a measure for each PLA element and relation, as it calculates the amount of products (ratio) that have a specific element or relation. The closer to one the better: a high **CRR** indicates that the element presents a good reuse rate. Values above 50% mean that the element was used at least in half of SPL products, what indicates a good reuse rate.

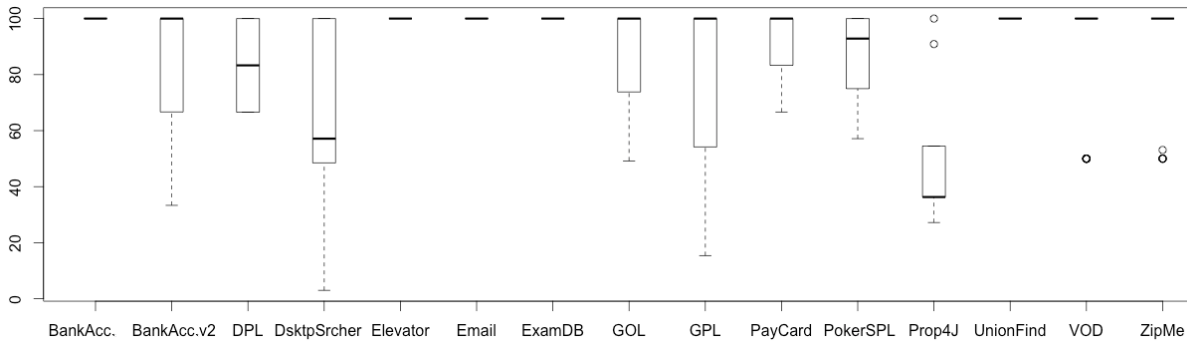


Figure 6.2 Boxplot of Component Reuse Rate per PLA

Figure 6.2 shows a boxplot with the distribution of the reuse rates for each SPL. The values range from 0 to 100, in which lower values indicate worse results in terms of reuse rate, and the higher values indicate ones.

6.3.2 Draw Product Line Results

Table 6.1 shows the average reuse rate for its recovered PLA. The **SSC** value indicates variability in 50% of the PLA elements, while the **RVC** value indicates variability in 2/3 of the PLA relations.

Table 6.2 CRR Measures for DPL elements

Element	CRR_{pair}	CRR_8	CRR_{all}
BasicRectangle	100.0	62.5	66.7
Canvas	100.0	100.0	100.0
Line	50.0	50.0	66.7
Main	100.0	100.0	100.0

Table 6.2 presents the **CRR** values for the PLA elements. The CRR_{pair} , CRR_8 and CRR_{all}

columns present the CRR value for recovery based on two products, eight products, and all SPL products configurations, respectively.

The CRR measures for *Canvas* and *Main* (two classes implementing common features) are 100%. For *BasicRectangle*, the CRR_{pair} measure is 100% and variability could not be identified. The CRR_8 decreased 35.5% (reaching 62.5%) and CRR_{all} increased 4.2% (reaching 66.7%). For *Line*, the CRR_{pair} measure is 50% and some variability could be identified. The CRR_8 remains 50% and CRR_{all} increased 16.7% – reaching 66.7%. DPL results do not confirm that the number of products used in the PLA recovery influences the precision of the CRR values for the PLA elements.

6.3.3 Video on Demand Results

The SSC value for VOD is 0.77, indicating that 32 products reused most of its PLA elements. We found similar results for the PLA relations (see Table 6.1).

The optional classes had a CRR of 50%, that is, these elements were used by half of the SPL products, indicating high component reuse inside the SPL. However, some relations presented a CRR of 25% and 3.25% indicating that few products used them. For this reason, the refactoring of the elements involved in this relation should be considered (ZHANG et al., 2008). Due to space limitation, information about the PLA and the Table with the CRR values for the 42 classes of the VOD SPL project are only available at the study website¹.

6.3.4 Zip Me Results

The SSC measure was 0.8 indicating the reuse of almost all the elements. The RSC was 0.7, also a high value for reuse of relations (see Table 6.1).

The PLA elements presented high CRR values – 25 common and 6 variable elements – above 50%. The PLA relations also presented high CRR values – 32 common and 14 optional relations. From the optional relations, 11 presented a CRR value above 50%; the other relations had a CRR of 25%.

6.3.5 Game of Life Results

The SSC value was 0.62%, indicating a larger amount of common elements rather than variable ones (see Table 6.1).

Although the GOL results indicated lower SSC values when compared to the VOD project, some elements were present in almost every GOL product (98% and 73%). In other words, the reuse rate for GOL is high. We found similar results for CRR and PLA relations.

Figure 6.3 shows the design structure matrix for GOL PLA. Rows and columns headers of a DSM are named after PLA elements. The darker items represent the commonalities of the PLA while the lighter items represent the variable elements. The tool colors the dependency between two common elements with blue, and between a variable element

¹<https://bit.ly/2UUMzCv>

and another element using red.

The GOL DSM shows that, unlike GPL and Prop4J, there is not a central node, *i.e.* an element that is related to almost every PLA element. The DSM also allows to visualize that PLA relations are scattered in the elements. We also noticed that some PLA elements did not have any relation; we believe this is due to features not implemented or discarded while their classes still remain in source code.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1 ButtonsToolBar																					
2 ClearGeneratorStrategy																					
3 FormGeneratorStrategy																					
4 GODLModel																					
5 GenerationScheduler																					
6 GenerationSelector																					
7 GeneratorStrategy																					
8 GolView																					
9 LifeForm																					
10 LifeFormIterator																					
11 Main																					
12 ModelObservable																					
13 ModelObserver																					
14 Playground																					
15 PlaygroundIO																					
16 PlaygroundMouseAdapter																					
17 PlaygroundPanel																					
18 PopUpMenu																					
19 RandomGeneratorStrategy																					
20 RuleSet																					
21 Suite																					

Figure 6.3 Design Structure Matrix for GOL

6.3.6 Graph Product Line Results

The SSC value was 0.6 indicating more common than variable elements. However, the opposite happened with the relations – the SSC value was 0.42. This scenario indicated that most of the PLA relations were variable. According to Zhang et al. (2008), this is a symptom of bad component reuse, suggesting a potential candidate for improvement.

The CRR values indicated that most variable elements presented a high reuse rate. However, we identified that some elements, such as *CycleWorkSpace* and *GlobalVarsWrapper*, presented a low CRR value. Furthermore, the majority of relations presented low CRR values. In other words, CRR values indicate that the PLA has flaws and bad quality (ZHANG et al., 2008). The elements *Graph* and *Vertex* have relationships with all the other elements.

6.3.7 Prop4J Results

Prop4J has no mandatory features, implementing only optional features. Metrics results (Table 6.1) reflect this characteristic. In addition, for the absence of mandatory features,

this SPL project allowed the generation of 5029 possible configuration of products based on optional features. For this reason, we instantiated 11 products using T-wise test configuration (HENARD et al., 2014). After PLA recovery based on a subset of 11 products, we identified only one common element (*Node*) and 13 variable elements. All the recovered relations are variable.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1 And														
2 AtLeast														
3 AtMost														
4 Choose														
5 Equals														
6 Implies														
7 Literal														
8 Node														
9 NodeReader														
10 NodeWriter														
11 Not														
12 Or														
13 Prop4JTest														
14 SatSolver														

Figure 6.4 Design Structure Matrix for Prop4J

Figure 6.4 shows the DSM for Prop4J PLA. The *Node* element is common to the 11 products analyzed. However, the relations were variable confirming the CRR values. These results suggest that the PLA has poor quality and that the SPL needs maintenance to improve the reuse rates.

In Prop4J DSM, we identified two central classes, *Node* and *Prop4J test* that relate to almost every class present in the PLA, which is an indicative of the God class smell (FOWLER, 2009) and require further investigation.

6.3.8 BankAccount and BankAccountV2 Results

BankAccount SPL implemented the variability in the source code. For this reason, the SSC and RSC values were 1, and SVC and RVC values were 0. On the other hand, BankAccountV2, the second version of the BankAccount SPL, introduced new features to the SPL.

The implementation of these new features required new classes and relations, which required the identification of variability in classes and relations that were absent from the previous version of this SPL. The BankAccountV2 SSC value was greater than 0.5, meaning that most of its elements are common to all products, what indicates a good PLA quality (ZHANG et al., 2008).

6.3.9 DesktopSearcher Results

The values for SSC and RSC were considered low (ZHANG et al., 2008). The SSC value was 0.26 which means that almost all of its classes are variable among the products. The CRR presented the same behavior. In most cases, the values were lower than 0.5, what indicates a need of improvement in the reuse rate.

Theses values indicated that the SPL products tend to have exclusive products that require specific features in only part of the products (APEL et al., 2013) which we believe to be the cause of the low reuse rate found in this SPL project.

6.3.10 PayCard Results

PayCard is a small-sized SPL project, with respect to the amount of product configurations, and classes. The project presents good reuse rates with a value of 0.71 for SSC, *i.e.*, most of its classes are common to all products. However, the dependencies among classes presented a value of 0.37 for RSC, which means that most of the relations are variable.

6.3.11 PokerSPL Results

PokerSPL is another small-sized SPL project, regarding the number of classes and products. It presented a 0.5 SSC value meaning that half of its classes are common to all products. The reuse rates could be better since most of the variability in this SPL was found on the values assumed by some of its classes attributes.

Accordingly, the CRR values for the SPL relations present similar results. A possible explanation is that most of the SPL variability is implemented in the values of class attributes.

6.4 DISCUSSION

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

6.4.1 Answers to the Research Questions

For the first research question, *RQ1. Does the number of products used in PLA recovery affect the precision of the PLA metrics?*, we verified if *The number of products used in PLA recovery affected the precision of the PLA metrics.*

Figure 6.5 shows SSC (dark grey color) and SVC (light grey color) projects' metrics collected during different stages of comparison. We started comparing two products and added the products gradually until all the products were considered.

We identified that the amount of information regarding the variability increased when we included more products in the comparison. Moreover, after a certain number of comparisons, the value of the metrics became constant. For instance, we compared 18 products aiming to recover all the variability details of the Zip Me SPL. We observed the same pattern on other SPL projects. For example, it was necessary to compare 17 products to recover all the variability details of the VOD SPL.

As the PLA recovery process examined and merged more products, the set of elements and relations that comprise the PLA and metrics values tends to stabilize. The set of products (after the metrics stabilization) had a common structure. We also identified this pattern when we analyzed the CRR values with different combination of products in the recovery. In other words, the sub-set of products has the same architecture.

Table 6.3 presents the CRR values from the Prop4J project. The CRR_{pair} , CRR_8 and CRR_{all} columns present the CRR value for recovery based on two products, eight products, and all SPL products, respectively. By comparing all the product, we identify the correct CRR values.

Table 6.3 CRR Measures for Prop4J

Element	CRR_{pair}	CRR_8	CRR_{all}
And	100.0	50.0	54.5
AtLeast	50.0	37.5	36.4
AtMost	50.0	37.5	36.4
Choose	50.0	37.5	36.4
Equals	100.0	37.5	27.3
Implies	100.0	50.0	54.5
Literal	100.0	87.5	90.9
Node	100.0	100.0	100.0
NodeReader	100.0	62.5	54.5
NodeWriter	50.0	75.0	54.5
Not	50.0	50.0	36.4
Or	50.0	37.5	36.4
Prop4JTest	50.0	37.5	36.4
SatSolver	50.0	37.5	45.4

Moreover, to answer the RQ1, we performed a correlation analysis among the variables of the exploratory study (see Figure 6.6). We identified a high correlation between the number of variants (in this analysis we considered products as variants) and the variables that address the variability (CO, OR, SVC, and number of optional features). Table 6.4 shows the *Spearman* correlation test that rejected the null hypothesis.

Table 6.4 Comparisons that rejected the null hypothesis RQ1

Comparison	p-value
Products-CO	7.0e-03
Products-OR	7.0e-03
Products-SVC	2.0e-02
Products-OF	2.0e-03

In the second research question, we used the ANOVA (Analysis of Variance) to test the variables and the p-value was 1.3e-07. Such evidence allows to reject the null hypothesis (H_{0a}) of equal population means. Therefore, it is possible to conclude that at least one PLA has reuse rate significantly different from the others.

To identify the different means, we applied the Tukey test. We performed and analyzed 105 comparisons, and only 12 of them presented statistically significant differences.

Table 6.5 shows p-values of the comparisons that rejected the null hypothesis (the SPLs involved in the test, and the p-value).

Table 6.5 Comparisons that rejected the null hypothesis RQ2

ID	Comparison	p-value
40	Elevator-DesktopSearcher	2.3e-03
43	GOL-DesktopSearcher	4.4e-02
49	VOD-DesktopSearcher	2.1e-03
50	ZipMe-DesktopSearcher	1.5e-03
57	Prop4J-Elevator	4.0e-05
74	Prop4J-ExamDB	1.9e-02
81	Propo4J-GOL	8.0e-04
92	Prop4J-PayCard	2.2e-02
96	Prop4J-PokerSPL	4.4e-02
100	UnionFind-Prop4J	1.9-e02
101	VOD-Prop4J	7.0e-05
102	ZipMe-Prop4J	4.0e-05

We also highlight the comparisons that rejected the null hypothesis in conformance with Table 6.5. Based on such data, we identified that Prop4J (in eight comparisons) and DesktopSearcher (in four comparisons) yielded worse reuse rates among the PLAs (see Figure 6.2).

We identified that optional features directly impact the reuse rate. The Prop4J presented the worst results because all their features were optional allowing the instantiation of 5029 products. Moreover, for the DesktopSearcher SPL, the 16 optional features allowed the creation of 462 products. These scenarios illustrate the complexity involved during an SPL project development. The variability management is a complex task that is also reflected in the PLA.

6.4.2 General Findings

Correlation between metrics. From the overall results for the fifteen SPL projects, we noticed that when the value of SSC was high, the PLA elements presented high CRR values as well. This may be a preliminary evidence for a correlation between SSC and CRR metrics.

Some metrics provided support for other metrics. The quantitative metrics CM and CO counted the number of mandatory and optional classes of the PLAs. They confirmed the SSC and SVC metrics values.

CM and CO provided support for the SSC, SVC, and CRR metrics. The former validated and confirmed the values of the latter.

Feature scattering and reuse rate. In projects with better metrics results (Zip Me, VOD, and GOL, respectively), the classes outnumbered the features, with feature scattering in a significant amount of classes. To perform this analysis, we verified how the feature selection to build each product was spread through the source code manually and compared to the metrics collected by the PLAR. The relation between feature scattering and high reuse rate deserves further investigation.

6.4.3 Threats to Validity

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

Internal Validity. PLAR Tool limitations may have impacted the results of the exploratory study. For instance, the input for PLAR Tool is a MDG file created by STAN4J and Analizo. Currently, the extracted MDG only supports “call” dependencies between modules. Inheritance relationships are not extracted.

External Validity. No industrial SPL Projects were used in this exploratory study; only open source SPL projects created for educational and research purposes were used. In order to minimize this threat we analyzed SPL projects from different domains.

Construct Validity. relacao ao estudo e os potenciais resultados (casa tambem com as hipoteses definidas); sera que foram definidas para refletir a realidade? ou ainda, sera que durante a execucao do estudo, ocorreu algo compativel com um bias, para ajustar os resultados às expectativas? The recovered PLA from the SPL projects were not verified by SPL developers. We contact them, but they rarely answer. To minimize this threat we performed a manual PLA extraction and compared to the results obtained by PLAR Tool which showed that the variable and common elements were mapped correctly.

6.5 CHAPTER SUMMARY

In this chapter, we presented the application of SAVaR and the results of an exploratory study to assess the quality of the recovered PLAs from 15 open source SPL projects implemented in Java. The PLAR Tool was used to support PLA recovery with the identification of commonality and variation points.

Eleven out of fifteen recovered PLAs had high quality according to the established criteria indicating high reuse of components during the SPL development phase. The results provided initial evidence regarding a correlation between the metrics values and the components reuse rate.

Some of our findings are: the number of products used in PLA recovery affected the variability identification; the number of optional features affected the components reuse rate; and there is a correlation between the metrics used to assess SPL reusability based on the recovered PLAs. These findings suggest that a minimum set of representative SPL products should be identified and selected for PLA recovery and that *component reuse rate* is a candidate metric for SPL reuse assessment.

Other contribution of the study presented in this chapter was the recovered architectural information for each SPL project, because none of them presented PLA documentation. The results of this exploratory study were also used to improve the design and execution of following empirical studies. In the next Chapter, we present the application of the SAVaR in the recovery of 10 PLAs and the definition of the PLA for the Apo-Games project.

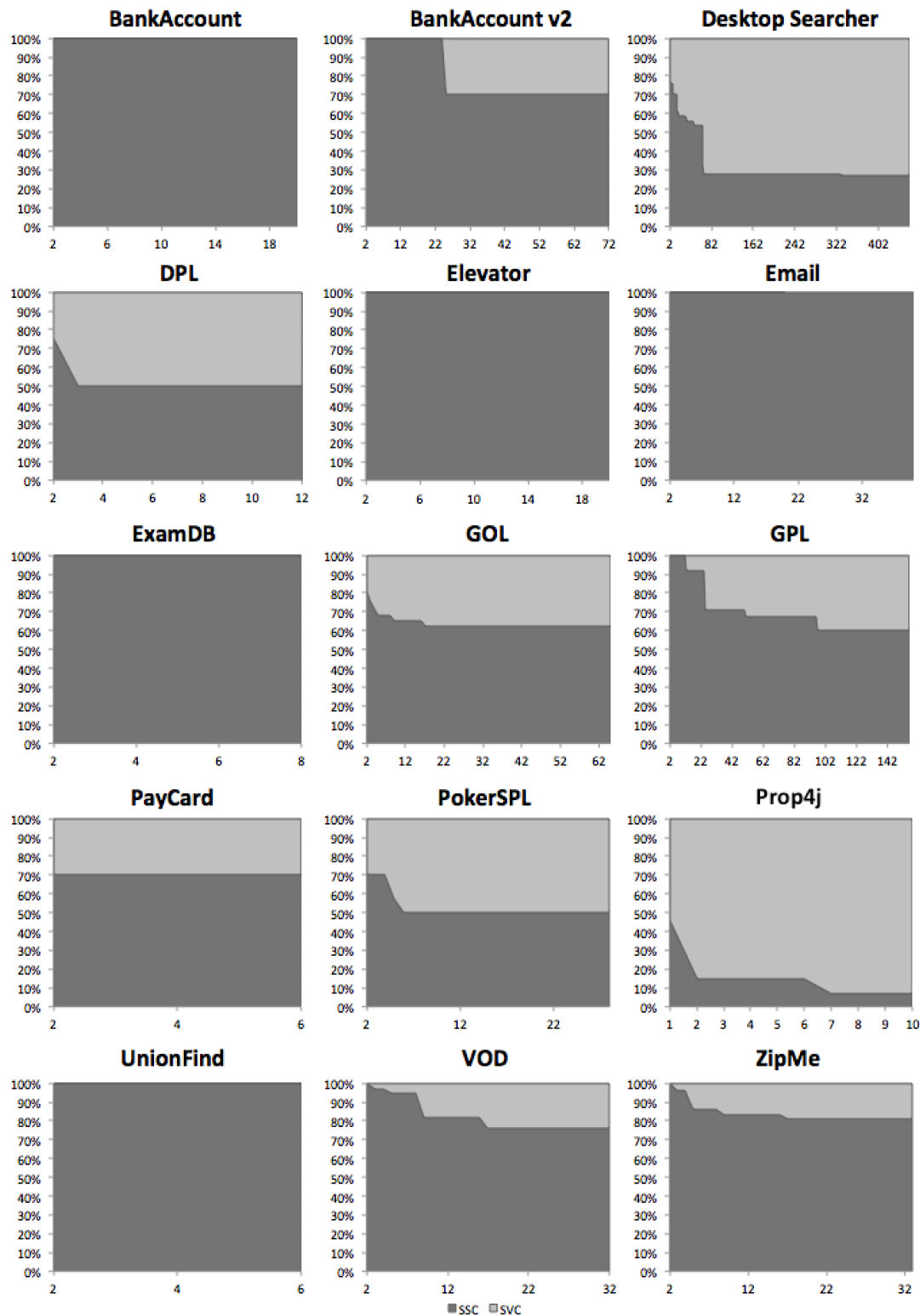


Figure 6.5 SSC and SVC metrics according to the number of products in the comparison

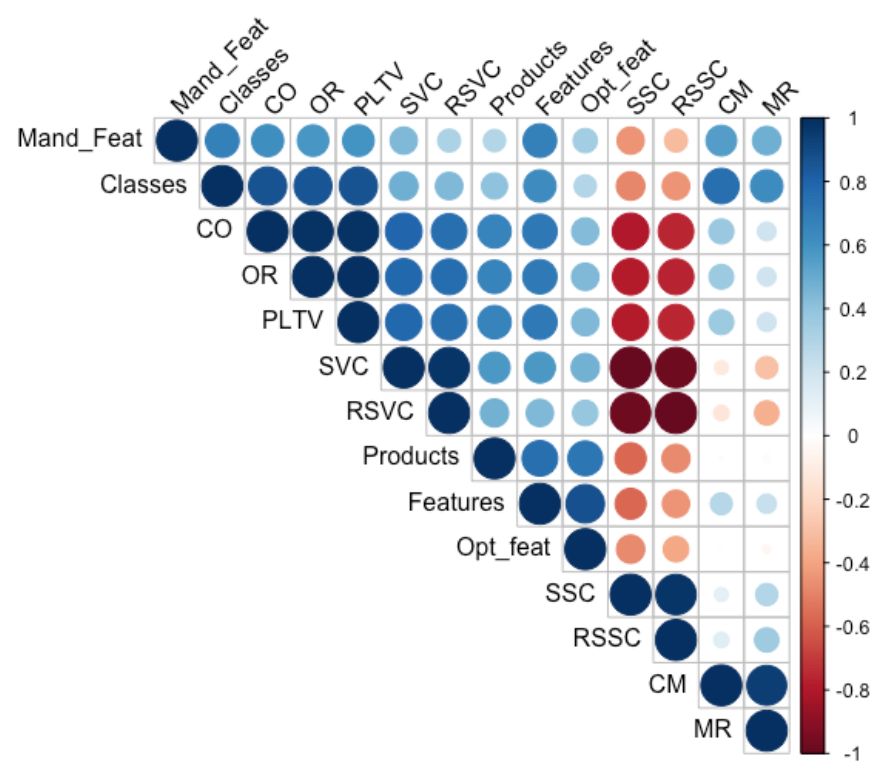


Figure 6.6 Correlation analysis

No man ever steps in the same river twice, for it's not the same river and he's not the same man. – Heraclitus

CASE STUDIES

Whenever the PLA documentation is missing, it can be recovered by reverse engineering the SPL variants. The recovered PLA is a relevant asset for both developers and architects, and can be used to drive specific activities of SPL development and evolution, such as understanding the SPL structure and its variation points, and assessing SPL reuse.

To evaluate **SAVaR**, we applied it in the case of the Apo-Games projects (LIMA et al., 2018, 2018). The Apo-Games¹ is a set of medium-sized games that have been implemented based on the clone-and-own approach. This case study has been proposed as a good candidate for research in the context of reverse engineering of variability (KRÜGER et al., 2018). Apo-Games evolved over time because of the inclusion of new games and adaptations. These games are composed of Java desktop and Android applications. Java desktop have evolved since 2006 until 2012. In 2012 the development of Android games started. In this work, we deal with the Java games of the Apo-Games repository². The games source code varies from 1.7 KLOC to 19.6 KLOC. They are medium-sized projects but cumulate to an overall size of 163.1 KLOC.

The experimentation in this real family of systems showed that **SAVaR** is able to identify variant outliers and help domain experts to take informed decisions to support PLA recovery. Moreover, we performed a study to investigate the use of **SAVaR** to recover the PLA for ten SPL projects. For each SPL and recovered PLA, reuse assessment was supported by existing reuse metrics.

The remainder of the Chapter is organized as follows. Section 7.1 describes the study on recovering the PLA of the Apo-games. Section 7.2 describes the PLA recovery of 10 open source projects. Section 7.3 draws concluding remarks.

¹<<http://apo-games.de>>

²<http://bitbucket.org/Jacob_Krueger/apogamessrc>

7.1 RECOVERING THE PLA OF THE APO-GAMES

Along this section, we investigated the application of SAVaR using the projects' legacy source code as input. The study focused on analyzing the source code of 20 projects developed using the Java programming language and 5 projects developed using the Android framework.

We next describe the evaluation design defined based on the Goal/Question/Metric (GQM) method (SOLINGEN et al., 2002).

Evaluation goal: *evaluating how SAVaR supports the cost-effective PLA recovery through the identification and removal of outliers.*

Research questions: Guided by our evaluation goal we derived the following research questions.

- *RQ1: How similar are the variants?* With this question we want to investigate, in one hand, how similar the variants are. On the other hand, we could also observe how different the variants are. Since our goal is to deal with outliers, we first need to figure out the degree of similarity and variability of the variants to determine which variant is too costly to be included in the PLA recovering process.
 - *Metric:* We relied on the analysis of the Jaccard similarity³. 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 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 variants' size (LOC) and the number of packages and classes exclusive to a variant. We used the Pearson correlation coefficient analysis.
- *RQ3: To what extent removing outliers from the analysis support recovering of better PLAs?* We aim to analyze the quality of the obtained solutions when removing outliers. The quality of such solutions is evaluated according to eight architectural metrics. Another point we take into account here is about 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 Chapter 2. We collected these metrics as a result of SAVaR. We applied the threshold analysis and collected the metrics after the threshold cut.

³<https://en.wikipedia.org/wiki/Jaccard_index>

Table 7.1 presents a summary of the GQM method for our evaluation.

Table 7.1 Describing the study according to GQM

Goal	Evaluating cost-effectiveness of SAVaR
Purpose	Analyze the impact of outliers
With respect to	product line architecture recovering
From the point of view	Researchers
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

Table 7.2 Apo-Games Projects – Metrics summary

Projects	Year	KLOC	#E	#Jr	#Ja	#T	#TJ	#P	#EP	%P	#C	#EC	%C
ApoDefense	2007	12917	-	✓	✓	-	✓	-	-	-	66	56	85%
ApoSkunkman	2007	8645	-	✓	-	✓	-	16	6	37%	-	-	-
ApoStarz	2008	6454	-	-	✓	✓	✓	11	1	9%	49	8	16%
ApoBot	2009	5857	-	-	✓	✓	✓	8	0	-	48	2	4%
ApoSoccer	2009	10736	-	✓	-	✓	-	18	10	55%	-	-	-
ApoCommando	2010	9820	✓	-	-	✓	✓	5	0	-	72	18	25%
ApoIceJumpR.	2010	8138	-	✓	-	✓	-	9	0	-	-	-	-
ApoPongBeat	2010	6591	✓	-	-	✓	✓	10	1	10%	79	31	39%
ApoIcarus	2011	5851	✓	-	-	✓	✓	9	0	-	59	15	25%
ApoMarc	2011	5493	-	-	✓	✓	✓	10	2	20%	59	6	10%
ApoMario	2011	17184	-	✓	-	✓	-	14	2	14%	-	-	-
ApoSlitherLink	2011	7313	✓	-	-	✓	✓	8	0	-	62	8	12%
ApoNotSoSimple	2011	7558	✓	-	-	✓	✓	10	0	-	57	1	2%
ApoRelax	2011	6688	✓	-	-	✓	✓	10	0	-	56	3	5%
ApoSimple	2011	19558	✓	-	-	✓	✓	16	6	37%	104	48	46%
ApoSnake	2012	6557	-	✓	-	✓	-	10	0	-	-	-	-
ApoSudoku	2012	5517	✓	-	-	✓	✓	9	0	-	41	2	5%
ApoImp	2012	6432	-	✓	-	✓	-	12	1	8%	-	-	-
Total	-	157309	8	7	4	17	12	185	29	15%	752	198	26%

Legend: [LOC] Lines of Code, [#E] projects available in Eclipse, [#Jr] projects available in Jar files, [#Ja] projects available in Java files, [#T] TGF files packages, [#TJ] TGF files 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

7.1.1 Case Study

To answer the research questions, we used the Apo-Games variants as input for SAVaR. From the 20 variants, we excluded three of them because they did not provide information for allowing the Information Extraction ②. Concretely, we identified that the developer did not use the packages structure in the implementation of the projects ApoCheating and Tutorvolley from 2006. Moreover, we eliminated ApoDefence variant of the packages analysis because the developer performed obfuscation to the source code and we did not have access to the original source code for this variant.

Table 7.2 presents the selected information of the variants. From the projects, eight were available as Eclipse projects, six are only jar files, and five presented only the Java

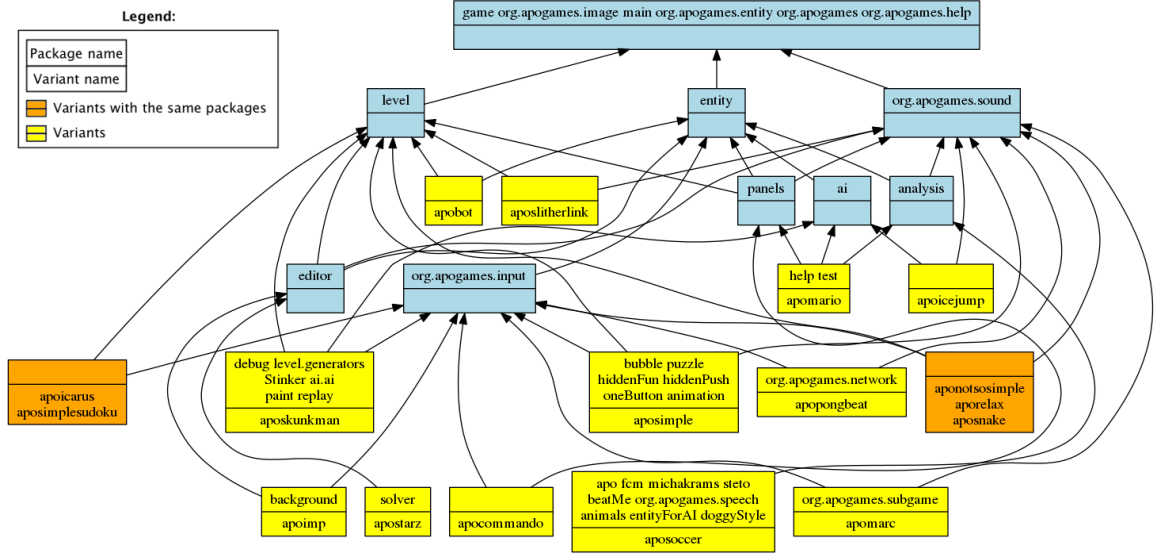


Figure 7.2 Excerpt of the Concept lattice with variants in the games

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 4 variants (ApoMario, ApoSoccer, ApoSimple and ApoSkunkman) which are dissimilar among them and among the rest of the variants. On the contrary, in the zoomed part, we can observe some variants which are very similar. As an extreme case, there are completely overlapped variants what indicates that certain games have the same PLA (e.g., ApoIcarus, ApoSimple and ApoSudoku).

We performed FCA to automatically obtain the representation shown in Figure 7.2 which is known as the pruned concept hierarchy (PETERSEN, 2004). The ApoBot variant in the upper half of the figure illustrates such a case. By recursively following the arrows until the root, we can know that ApoBot 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 previously mentioned variants ApoIcarus, ApoSimple 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 another variant). Both Figures 7.1 and 7.2 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.

Figures 7.3 and 7.4 show the correlation between Lines of Code (LOC) and exclusive variants' information (packages and classes implemented for a specific variant). Figure 7.3 shows the correlation between exclusive packages and the size of the variants and Figure 7.4 shows the correlation between exclusive classes and the variants' size. The correlation between exclusive packages and variants LOC is 0.51. Moreover, the correlation between exclusive classes and variants' size is even stronger (0.80).

Table 7.3 Recovered Metrics from the PLAs (Packages)

TH	SSC	SVC	RSC	RVC	CO	OR	CM	MR
00%	0.15	0.85	0.01	0.99	35	159	6	3
M. 1st	0.30	0.70	0.03	0.97	14	85	6	3
M. 2nd	0.50	0.50	0.07	0.93	6	37	6	3
06%	0.43	0.57	0.05	0.95	8	57	6	3
12%	0.55	0.45	0.08	0.92	5	37	6	3
18%	0.60	0.40	0.10	0.90	6	33	6	3
65%	0.70	0.30	0.30	0.70	3	7	6	3
77%	0.85	0.15	0.43	0.57	1	4	6	3
95%	1.00	0.00	1.00	0.00	0	0	6	3

Legend: [TH] Threshold, [SSC] Structure Similarity Coefficient, [SVC] Structure Variability Coefficient, [RSC] Relation Similarity Coefficient, [RVC] Relation Variability Coefficient, [CO] PackageOptional, [OR] OptionalRelation, [CM] PackageMandatory, [MR] MandatoryRelation

Table 7.4 presents the collected metrics for classes analysis. We executed SAVaR 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 values of threshold. We believed 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 high number of exclusive classes to raise the number of mandatory classes and improve SSC and SVC metrics' values.

Table 7.5 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 high number of exclusive classes, the SSC and SVC metrics did not change. We identified metrics improvements when we applied the threshold.

7.1.4 Discussion

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

7.1.5 RQ1 - How similar the variants are

The Jaccard similarity measure indicated that four variants can be considered as outliers, namely ApoSoccer, ApoSimple, ApoSkunkman, and ApoMario, because they are dissim-

Table 7.4 Recovered Metrics from the PLAs (Classes)

TH	SSC	SVC	RSC	RVC	CO	OR	CM	MR
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

Legend: [TH] Threshold, [SSC] Structure Similarity Coefficient, [SVC] Structure Variability Coefficient, [RSC] Relation Similarity Coefficient, [RVC] Relation Variability Coefficient, [CO] ClassOptional, [OR] OptionalRelation, [CM] ClassMandatory, [MR] MandatoryRelation

Table 7.5 PLA Metrics after eliminating some variants and Threshold analysis (Classes)

EV	SSC	SVC	RSC	RVC	CO	OR	CM	MR
0	0.01	0.99	0.00	1.00	278	625	2	0
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

Legend: [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, [MR] MandatoryRelation

ilar 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 ApoMario = 14%). We could not investigate the classes from ApoSoccer, ApoSkunkman, and ApoMario as the Java files were not available. However, we believe they followed the same pattern as ApoSimple with a high number of exclusive classes (46% of the classes).

7.1.6 RQ2 - Correlation between variants' size and likely outliers

The data reveal that for the Apo-Games case study, the differentiation among variants 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).

7.1.7 RQ3 - Impact of outliers removal in the recovery of better PLAs

In this analysis, we considered the results of applying the threshold technique. The implementation of this technique allowed the reduction of the number of exclusive packages and classes without removing the outliers 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 believed this behavior happened because it is easier to maintain the organization in the packages than in the classes. For this reason, we eliminated four outliers 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.

7.1.8 Threats to Validity

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

7.1.8.1 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 tools limitation and unavailability of the source code of some projects, we cannot extract the classes information of all the variants.

7.1.8.2 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, this study can help in building evidence regarding the impact of the variability in the context of PLA recovery. Moreover, another issue we found is related to packages and classes implementing the same logic but, using different names. We eliminated information specific to projects to reduce this issue impact.

7.1.8.3 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.

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 metrics analysis and PLA representation. Even though, it still impossible to eliminate human intervention.

7.1.8.4 Conclusion Validity.

Low statistical power. One threat to this study was the sample size. From the 20 variants, we considered 17 in the package analysis and only 12 in the classes analysis. However, as the purpose was to provide 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. To mitigate this treat, we performed the study in Section 7.2.

7.2 RECOVERING THE PLA OF 10 OPEN SOURCE PROJECTS

This section describes the application of the PLA recovery in the context of ten open source SPL projects from different domains were selected for this study, based on the following criteria: lack of documented PLA and source code written in Java (a constraint imposed by STAN4J).

The selected SPL projects are:

- *DesktopSearcher* - An SPL that implements programs for indexing and content based searching in files;
- *Game of Life (GOL)* - An SPL that simulates the board game - game of life;
- *Graph Product Line (GPL)* - An SPL for implementing graph manipulation libraries.
- *Health Watcher* - A real web-based system information developed to improve the quality of the services provided by health care institutions (GREENWOOD et al., 2007);

Table 7.6 Describing the study according to GQM

Goal	Evaluating cost-effectiveness of SAVaR
Purpose	Analyze the impact of exclusive variable elements
With respect to	PLA recovering
From the point of view	Researchers
In the context of	Extraction from 10 open source projects
Question	Metrics
RQ4	SSC, SVC, RSC, RVC

- *MobileMedia* - An SPL for manipulating photo, music, and video on mobile devices, such as mobile phones (FIGUEIREDO et al., 2008);
- *Prop4J* - An SPL for arbitrary propositional formulas;
- *Message* - An SPL that implements Instant Message products;
- *Video on Demand (VOD)* - An SPL for video-on-demand streaming applications;
- *Webstore* - An SPL that implements an online store;
- *Zip Me* - An SPL for file compression software.

We describe the next SAVaR evaluation using the Goal/Question/Metric (GQM) approach (SOLINGEN et al., 2002). Table 7.6 presents a summary of the GQM method for our evaluation.

Evaluation goal: *evaluate how SAVaR and guidelines support the cost-effective PLA recovery through the identification and removal of exclusive variable elements.*

Research question:

- *RQ4: To what extent the elimination of exclusive variable elements can support the recovering of a better PLA?*

With this research question we focus on the quality of the recovered PLA when removing exclusive elements, that is, variable elements that appear in a small percentage of variants. The quality of such PLA is quantitatively evaluated according to architectural metrics. Another point we take into account is about the implementation level. We want to investigate the impact of the removal of exclusive elements at the class level.

Metrics: To answer RQ4, we used the SSC, SVC, RSC, and RVC metrics. We applied the threshold analysis and collected the metrics after the threshold cut.

Table 7.7 presents raw data from the 10 open source projects selected for our empirical evaluation. We selected projects with different number of variants (#V), number of classes (#C), average number of classes per project (avg), and strategies for variants generation (Gen.). Moreover, we used different types of variants generation and projects that provided the variants to verify if SAVaR support them.

Table 7.7 Analyzed Projects

Projects	#V	#C	avg	#E	#R	#P	Gen.
Desktop Searcher	462	18942	41	12504	30126	10	AH
GOL	64	1344	21	1197	1998	4	NA
GPL	156	2340	15	1843	4341	7	CD
Health Watcher	10	1396	136	1113	4857	11	NA
MobileMedia	8	346	43	243	406	8	NA
Prop4j	452	6328	14	3648	6710	10	FH
SPL Message	10	680	68	493	743	5	Ant
VOD	32	1344	42	1184	2082	3	NA
Webstore SPL	10	710	71	534	1408	4	Ant
ZipME	31	992	32	897	1226	4	NA
Total	1226	33783	483	23746	53897	66	-

Legend: [#V] Number of variants, [#C] Total number of classes analyzed, [avg] average number of classes per project, [#E] Total number of elements analyzed, [#R] Total number of relations analyzed, [#P] Number of execution of SAVaR, [Gen.] Variants generation, [AH] AHEAD, [NA] Not Available, [CD] CIDE, [FH] FeatureHouse

We used the FeatureIDE (MEINICKE et al., 2016) to generate the variants from AHEAD, CIDE, and FeatureHouse composers. Projects implemented with `#ifdefs` used Ant build for variants generation. We used the other projects variants source code available in the projects' repositories.

7.2.1 Preparation

We collected information about the projects and downloaded the source code and other assets from the repository. We identified the mechanism used to implement the variability because the selection of recovery techniques and extraction tools depends on them (Section 4.3.1).

Then, we extracted each variant structural information. We performed the variability identification. We mapped the mandatory elements that were implemented in all the variants and the variable elements that were implemented in only some variants.

With the recovered PLA outputs, we analyzed the metrics and reports. These outputs were used to suggest improvements to the results. We collected the elements implementation frequency to define the threshold values.

Based on the threshold, we performed the PLA recovery again. In this way, we provided a set of recovered PLAs allowing practitioners (architects and developers) to select the PLA according to their interests.

7.2.2 Analysis and Interpretation

Table 7.8 presents the collected metrics for PLA recovery analysis within the threshold results. We ran the SAVaR according to threshold values based on a report generated by SAVaR. The report identifies the elements according to their existence in the variants.

For instance, when a class is implemented in all the variants, the report informs that this class appears in the implementation of 100% of the variants. The complete experimental setting and results can be found at the website⁴.

7.2.2.1 Answering RQ4

To answer the research question RQ4, we analyzed the threshold technique results. The implementation of this technique allowed the reduction of the number of variable classes without eliminating variants. The technique provides an alternative to the solution we proposed in our previous study (LIMA et al., 2018). Instead of identifying and eliminating outliers (variant that introduces a high number of variable elements that are implemented in only that variant), we keep all the variants during the analysis.

Projects such as `GOL`, `VOD`, `Webstore`, and `ZipME` allowed a small number of executions of `SAVaR` because the `SSC` and `SVC` values were balanced. Such balance may indicate that these projects considered the variability impact upfront during the development phase.

Moreover, the threshold technique allowed us to improve some projects, metrics such as the `MobileMedia` results. In this case, the report identified that the majority of the variants implemented some classes such as `AlbumData` (87%), `AddPhotoAlbum` (76%), and `ImageAccessor` (75%). In other words, this evidence may point out that, during SPL evolution, stakeholders should consider the modification of features and these classes to *mandatory*.

Projects with high value of `SVC` and `RSC` (near to 1) could lead to variability explosion. For instance, `Prop4j` project allows the creation of 4100 variants. It is hard to maintain and propagate the changes in an evolution scenario. In order to support this issue, the threshold technique improved the results slightly even with a 70% in information reduction. Our report identified that the majority of the variants implemented two classes: `Literal` (99%) and `SatSolver` (98%). By considering these classes as mandatory, the values for these metrics improved.

The metrics (high value of `SSC` and `RSC`) indicates that `GOL`, `VOD`, `Webstore`, `ZipME`, `GPL`, `Message` projects variability can be improved. On the other hand, projects with a high value of `SVC` and `RSC` (e.g. `Desktop Searcher`, `MobileMedia`, and `Prop4j`) indicate that improvements in the definition of mandatory elements are necessary.

We identified that the information reduction provided by the threshold technique allowed the balance of `SSC` and `RSC` metrics, and `RSC` and `RVC` metrics. It is relevant to support and raise the abstraction level in architectural level. Moreover, in some cases, the technique reduced the information up to 70% and provided the metrics' balance.

7.2.3 Threats to Validity

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

⁴<<https://bit.ly/2RYwfU4>>

7.2.3.1 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 some cases, 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 implemented the threshold technique.

7.2.3.2 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 selected projects represent a small portion of the domain we want to generalize. However, it is one more case that can help in the evidence building regarding the impact of the variability in the context of PLA recovery. Moreover, another issue we found is related to classes implementing the same logic but, using different names. We eliminated information specific to projects to reduce this issue impact and we selected variants in projects developed by the same team.

7.2.3.3 Construct Validity

Mono-operation bias. We only considered subjects of the open source projects. 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.

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 effectiveness of PLA recovery and improvement of the recovered PLA. We based our analysis on metrics analysis and PLA representation. Even though, we cannot reject stakeholders' influence.

7.2.3.4 Conclusion Validity.

Low statistical power. The main threat to this study was the sample size. From the 1226 variants, we focused on the classes analysis and their relationships. SAVaR also supports packages and files abstraction. However, since the purpose of this study was to provide initial evidence on how the threshold improves the effectiveness of the PLA recovery, we understand that for generalizing such findings we need a larger sample.

7.3 CHAPTER SUMMARY

PLA recovery can provide useful information for defining the foundations to facilitate SPL adoption. Instead of working from scratch, the recovered PLA can support the development and maintenance tasks by providing a starting point. In this context, one of the main issues is to manage the variability introduced by some variants (*i.e.* outliers).

In this chapter, we presented the application of SAVaR and the results of two empirical studies to assess the quality of the recovered PLAs from 10 open source SPL projects.

We also applied **SAVaR** to recover the Apo-Games PLA. We performed a formal concept analysis to identify the outliers. We implemented the threshold analysis to reduce the number of exclusive components without eliminating the variants of the recovered PLA. Next chapter, we present the conclusion and future directions of this thesis.

Table 7.8 Recovered Metrics from the PLAs

TH	SSC	SVC	RSC	RVC	CO	OR	CM	MR	%rd
Desktop Searcher									
00%	0.27	0.73	0.09	0.91	30	134	11	14	-
04%	0.29	0.71	0.11	0.89	28	118	11	14	11%
25%	0.30	0.70	0.16	0.84	25	72	11	14	40%
29%	0.32	0.68	0.18	0.82	24	66	11	14	45%
37%	0.33	0.67	0.19	0.81	22	58	11	14	51%
43%	0.35	0.65	0.20	0.80	21	55	11	14	54%
49%	0.52	0.43	0.38	0.62	10	23	11	14	80%
58%	0.61	0.39	0.42	0.58	7	19	11	14	84%
97%	1.00	0.00	1.00	0.00	0	0	11	14	100%
GOL									
00%	0.62	0.38	0.69	0.31	8	11	13	24	-
50%	0.76	0.24	0.80	0.20	4	6	13	24	47%
74%	0.86	0.14	0.88	0.12	2	3	13	24	73%
99%	1.00	0.00	1.00	0.00	0	0	13	24	100%
GPL									
00%	0.60	0.40	0.41	0.59	6	23	9	16	-
16%	0.64	0.36	0.43	0.57	5	22	9	16	07%
39%	0.69	0.31	0.48	0.52	4	17	9	16	28%
47%	0.80	0.20	0.59	0.41	2	11	9	16	55%
62%	0.90	0.10	0.89	0.11	1	2	9	16	90%
81%	1.00	0.00	0.10	0.00	0	0	9	16	100%
Health Watcher									
00%	0.42	0.58	0.29	0.71	91	550	66	233	-
11%	0.49	0.51	0.32	0.68	67	501	66	233	11%
21%	0.54	0.46	0.38	0.62	55	367	66	233	34%
31%	0.56	0.44	0.40	0.60	51	360	66	233	36%
41%	0.58	0.42	0.51	0.49	46	230	66	233	57%
51%	0.61	0.39	0.53	0.47	42	200	66	233	62%
61%	0.63	0.37	0.62	0.38	39	147	66	233	71%
71%	0.64	0.36	0.66	0.34	37	119	66	233	76%
81%	0.73	0.27	0.81	0.29	25	53	66	233	88%
91%	1.00	0.00	1.00	0.00	0	0	66	233	100%
MobileMedia									
00%	0.12	0.88	0.02	0.98	52	148	7	3	-
13%	0.15	0.85	0.03	0.97	41	94	7	3	32%
26%	0.20	0.80	0.05	0.95	29	54	7	3	58%
38%	0.25	0.75	0.08	0.92	22	39	7	3	69%
51%	0.30	0.70	0.10	0.90	17	26	7	3	78%
76%	0.41	0.59	0.30	0.70	10	6	7	3	92%
88%	1.00	0.00	1.00	0.00	0	0	7	3	100%
Prop4j									
00%	0.07	0.93	0.00	1.00	13	67	1	0	-
01%	0.08	0.92	0.00	1.00	12	49	1	0	24%
42%	0.09	0.91	0.00	1.00	11	13	1	0	70%
48%	0.10	0.90	0.00	1.00	10	12	1	0	72%
50%	0.12	0.88	0.00	1.00	7	9	1	0	80%
51%	0.20	0.80	0.00	1.00	4	6	1	0	87%
55%	0.25	0.75	0.00	1.00	3	5	1	0	90%
65%	0.34	0.66	0.00	1.00	2	4	1	0	92%

Legend: [TH] Threshold, [SSC] Structure Similarity Coefficient, [SVC] Structure Variability Coefficient, [RSC] Relation Similarity Coefficient, [RVC] Relation Variability Coefficient, [CO] Class Optional, [OR] Optional Relation, [CM] Class Mandatory, [MR] Mandatory Relation, [%rd] Percentage of reduced variable elements per threshold cut

Continued on next page...

A journey of a thousand miles begins with a single step – Lao-Tsé

CONCLUSION

This thesis presented the **SAVaR**, a PLA recovery approach. In this investigation, our main efforts were mainly focused on understanding how to recover and manage the variability at the architectural level. The main goal was to achieve PLA recovery in a systematic way.

In this way, we performed a SMS on the relationship between SAR and PLA (Chapter 3). Moreover, we verified the possibility to adapt the existing solution proposal.

The evidence raised in Chapter 3 provided insights to create the **SAVaR** approach (Chapter 4) that considered the tasks to perform the PLA recovery. We also identified that the existing approaches did not explain how to apply the PLA recovery in details. As a result, we described all the steps to support PLA recovery and created guidelines to lead the recovery efficiently. We identified a gap regarding the lack of reuse of the existing solution. **SAVaR** focused on the reuse of the existing solution proposal.

In Chapter 5, we performed an exploratory study to evaluate the approach. We applied it in the context of SPL projects. Then, we gathered the subjects, feedback to improve **SAVaR** based on their experience. Chapter 6 and Chapter 7 describe the empirical studies we performed after the improvements and suggestions raised in Chapter 5. In Chapter 6, we recovered the PLA of 15 open source SPL projects by applying the proposed approach. Finally, Chapter 7 describes the application of the PLA recovery approach on the Apo Games project and on 10 open source SPL projects.

8.1 RELATED WORK

Wu et al. (2011) presented a semi-automatic PLA recovery approach. The authors defined measures to detect similarity and variability points in software products, source code of the same domain in order to migrate to the SPL paradigm. The study reports on a case study carried out with an industrial product line. The assumption is that legacy products of a same domain have similar designs and implementation that can be used to build a SPL. In **SAVaR**, we build the PLA from the products generated by the SPL project.

Losavio et al. (2013) proposed a reactive refactoring bottom-up process to build a PLA from existing similar software product architectures of a domain. The main assets were expressed by UML logical views. Their work is focused on the construction and representation of a candidate PLA followed by an optimization process to obtain the final PLA. The refactoring process was applied to a case study in the robotics industry domain. The focus of our work is the assessment of recovered PLA based on reuse metrics.

8.2 CONTRIBUTIONS

The contributions of this doctoral work are:

- We performed a *SMS* to investigate SAR and PLA relationship. The study provided an overview of the state-of-the-art. It allowed the identification of gaps and opportunities for developing the research.
- We proposed **SAVaR**, an *approach to PLA recovery from variants' source code that includes guidelines to aid the recovery process*. The PLA recovery approach documents a set of guidelines to recover variability-aware development views from variants' source code.
- We proposed and implemented *two techniques for variability identification at the architectural level*. The recovered variability information from SPL projects, together with its representation at the architectural level provide up-to-date structural PLA documentation, synchronized with SPL source code, that can be useful for SPL stakeholders to perform their tasks.
- We proposed and implemented a *threshold technique to tame variability explosion and improve the recovered PLA results*. The technique leverages the reduction of the variability in the recovered PLA while keeping all the available variants in the analysis. It provides a set of outputs that can be selected according to stakeholders' interests.
- We evaluated the **SAVaR** approach by applying it in different contexts and scenarios. We based on empirical software engineering methods and provided information for further replications.

8.3 RESEARCH LIMITATIONS

There are some limitations we came across while developing this thesis. They are presented next.

- *Dependency of tools and lack of support for large-scale projects*: we depend on some tools (both from academia and industry) such as Stan4J to propose examples of tool chain. However, these tools still lack support for SPL projects and variability identification. For this reason, we developed a tool to support **SAVaR**. We identified that the tools' majority did not support large scale systems;

- *Manual definition of threshold values:* we identified all the values possible and execute the PLA recovery (automatically) for each value;
- *Bring the variability identification to higher levels of abstraction:* in the case of object-oriented projects, we analyze the projects' packages and classes. In the case of procedural projects (large scale projects i.e. highly configurable systems), we analyzed the projects, files and functions;
- *Static representation of the PLA:* Due to scope limitation, we focused on the static analysis of the recovered PLA. We used bottom-up processes based on the variants' source code.

8.4 FUTURE RESEARCH DIRECTIONS

We identified some future directions that might be the focus of further research. Among them, we can mention:

- *Development of new tools and extended tool chain examples:* we found some gaps that can be addressed by the development of new tools. During the research we developed some converters to allow the transformation of a tools' output to other tools' input. However, there is work to be done to allow the integration of new tools. Moreover, we believe that extending the tool chains can bring contribution for recovering the variability in architectural level;
- *Create a new representation to raise the abstraction level of SAVaR:* a future research opportunity is raising the abstract level of SAVaR and provide support for the variability identification in this higher architectural level;
- *Interactive visualization of the recovered information:* we believe that improve the SAVaR output visualization by developing an interactive edition of the recovered PLA and integrate the different outputs;
- *Perform the PLA recovery with dynamic analysis tools and techniques:* another research direction is the introduction of dynamic analysis tools and techniques to improve the variability identification during the variants execution;
- *Creation of a decision model to support the PLA recovery:* Recovering the PLA of a project is not a simple task. For this reason, we proposed the guidelines. However, we believe that by combining the guidelines with a decision model, it could make accessible to subjects without experience in recovering architecture. Moreover, it can drive the research to the next level which is the full automation of the PLA recovery;
- *Introduce Search Based Software Engineering (SBSE) in the definition of the threshold values:* A future step towards the full automation of the recovery process is the use of SBSE to allow the automatic definition of the threshold values;

- *Utilize the recovered PLA to identify possible design smells and suggest refactoring recommendations:* based on SAVaR outputs, the recovered information can lead to suggestions of improvement and design smell identification;
- *Consider different versions of a project:* by using SAVaR and considering each version as a variant it can allow to understand how the versions evolved;
- *Experimentation with more case studies:* another research direction is the evaluation of SAVaR using more case studies, for instance by recovering the PLA of more complex open source projects. This would allow evaluating SAVaR scalability. Moreover, survey with SPL experts can suggest improvements for the approach steps;
- *Deeper analysis regarding feature implementation:* investigate how the implementation of features propagates over the source code and use SAVaR to support the investigation process.

8.5 CONTRIBUTIONS SO FAR

The knowledge developed during this work have been published. Table 8.1 shows published papers. Others were submitted to relevant conferences and journals of the field.

Table 8.1 SPL Projects

Paper Title	Venue	Part.
Related topics publications		
Software Architecture Documentation for Developers: A Survey (ROST et al., 2013) – Conference Paper	ECSA’13	<i>Minor</i>
Unveiling Architecture Documentation: Brazilian Stakeholders in Perspective (LIMA-NETO et al., 2015) – Technical Report	PGCOMP’15	<i>Major</i>
Thesis related publications		
Characterizing Product Line Architecture Recovery (LIMA-NETO; CHAVEZ; ALMEIDA, 2014) – Extended Abstract	CBSOFT’14	<i>Significant</i>
Initial Evidence for Understanding the Relationship between Product Line Architecture and Software Architecture Recovery (LIMA-NETO et al., 2015) – Conference Paper	SBCARS’15	<i>Significant</i>
A Systematic Review on Metamodels to Support Product Line Architecture Design (LIMA; CHAVEZ, 2016) – Conference Paper	SBES’16	<i>Significant</i>
Product Line Architecture Recovery: An Approach Proposal (Extended Abstract) (LIMA, 2017) – Doctoral Symposium	ICSE’17	<i>Significant</i>
Investigating the Recovery of Product Line Architectures: An Approach Proposal (LIMA; CHAVEZ; ALMEIDA, 2017) – Doctoral Symposium	ICSR’17	<i>Significant</i>
PLAR Tool – A Software Product Line Architecture Recovery Tool (CARDOSO et al., 2017b) – Tool Paper	CBSoft Tools’17	<i>Major</i>
Investigating the Variability Impact on the Recovery of Software Product Line Architectures: An Exploratory Study (CARDOSO et al., 2017a) – Conference Paper	SBCARS’17	<i>Significant</i>
Recovering the Product Line Architecture of the Apo-Games – Challenge Track (LIMA et al., 2018)	SPLC’18	<i>Significant</i>
Towards an Automated Product Line Architecture Recovery: The Apo-Games Case Study – Conference Paper (LIMA et al., 2018)	SBCARS’18	<i>Significant</i>
Product Line Architecture Recovery with Outlier Filtering in Software Families: The Apo-Games Case Study	JBCS’19	<i>Significant</i>

REFERENCES

- ABELE, A.; JOHANSSON, R.; LÖNN, H.; PAPADOPOULOS, Y.; REISER, M.-O.; SERVAT, D.; TÖRNGREN, M.; WEBER, M. The cvm framework : A prototype tool for compositional variability management. In: *Proceeding of : Fourth International Workshop on Variability Modelling of Software-Intensive Systems*. [S.l.: s.n.], 2010. p. 101–105.
- AHMED, F.; CAPRETZ, L. F. The software product line architecture: An empirical investigation of key process activities. *Inf. Softw. Technol.*, Butterworth-Heinemann, v. 50, n. 11, p. 1098–1113, 2008.
- ALBAUM, G. The Likert scale revisited: an alternate version.(product preference testing). *Journal of the Market Research Society*, v. 39, n. 2, p. 331–343, 1997.
- ANGELOV, S.; GREFFEN, P.; GREEFHORST, D. A framework for analysis and design of software reference architectures. *Information and Software Technology*, v. 54, n. 4, p. 417–431, 2012.
- ANGERER, F.; PRÄHOFER, H.; LETTNER, D.; GRIMMER, A.; GRÜNBACHER, P. Identifying inactive code in product lines with configuration-aware system dependence graphs. In: *Proceedings of the 18th International Software Product Line Conference - Volume 1*. [S.l.]: ACM, 2014. p. 52–61.
- APEL, S.; BATORY, D.; KASTNER, C.; SAAKE, G. *Feature-Oriented Software Product Lines*. [S.l.]: Springer, 2013.
- BASS, L.; CLEMENTS, P.; KAZMAN, R. *Software Architecture in Practice*. [S.l.]: Addison-Wesley Longman Publishing Co., Inc., 2003.
- BATORY, D. A tutorial on feature oriented programming and product-lines. In: *Proceedings of the 25th International Conference on Software Engineering*. [S.l.]: IEEE Computer Society, 2003. p. 753–754.
- BEUCHE, D. pure::variants. In: _____. *Systems and Software Variability Management: Concepts, Tools and Experiences*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. p. 173–182.
- BOSCH, J. Product-line architectures in industry: a case study. In: *Proceedings of the 1999 International Conference on Software Engineering*. [S.l.: s.n.], 1999. p. 544–554.
- BOSCH, J.; CAPILLA, R. Variability implementation. In: _____. *Systems and Software Variability Management: Concepts, Tools and Experiences*. [S.l.]: Springer Berlin Heidelberg, 2013. p. 75–86.

- BOSCH, J.; FLORIJN, G.; GREEFHORST, D.; KUUSELA, J.; OBBINK, J. H.; POHL, K. Variability issues in software product lines. In: *Revised Papers from the 4th International Workshop on Software Product-Family Engineering*. [S.l.]: Springer-Verlag, 2002. p. 13–21.
- BRERETON, P.; BUDGEN, D.; KITCHENHAM, B. A. *Evidence-based software engineering and systematic reviews*. [S.l.]: CRC Press, 2016. (Chapman & Hall/CRC innovations in software engineering and software development). ISBN 978-1-4822-2866-3, 199-201-203-2, 1482228661.
- BUDGEN, D.; TURNER, M.; BRERETON, P.; KITCHENHAM, B. Using Mapping Studies in Software Engineering. In: *Proceedings of PPIG 2008*. [S.l.]: Lancaster University, 2008. p. 195–204.
- BUSCHMANN, F.; MEUNIER, R.; ROHNERT, H.; SOMMERLAD, P.; STAL, M. *Pattern-Oriented Software Architecture - Volume 1: A System of Patterns*. [S.l.]: Wiley Publishing, 1996. ISBN 0471958697, 9780471958697.
- CARDOSO, M. P. S.; LIMA, C.; ALMEIDA, E. S. de; MACHADO, I. do C.; CHAVEZ, C. von F. G. 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, 2017. (SBCARS '17), p. 12:1–12:10. ISBN 978-1-4503-5325-0. Available at: <<http://doi.acm.org/10.1145/3132498.3133835>>.
- CARDOSO, M. P. S.; LIMA, C.; CHAVEZ, C.; MACHADO, I. do C. PLAR Tool – A Software Product Line Architecture Recovery Tool. In: *8th Brazilian Conference on Software: Theory and Practice - Tool Session*. [S.l.: s.n.], 2017.
- CHEN, L.; BABAR, M. A. A systematic review of evaluation of variability management approaches in software product lines. *Inf. Softw. Technol.*, Butterworth-Heinemann, v. 53, n. 4, p. 344–362, 2011.
- CHEN, L.; BABAR, M. A.; CAWLEY, C. A status report on the evaluation of variability management approaches. In: *13th International Conference on Evaluation and Assessment in Software Engineering*. [S.l.]: BCS, 2009.
- CHIKOFSKY, E. J.; CROSS, J. H. Reverse engineering and design recovery: a taxonomy. *IEEE Software*, v. 7, n. 1, p. 13–17, 1990.
- CLEMENTS, P.; BACHMANN, F.; BASS, L.; GARLAN, D.; IVERS, J.; LITTLE, R.; MERSON, P.; NORD, R.; STAFFORD, J. *Documenting Software Architectures: Views and Beyond (2nd Edition)*. 2. ed. [S.l.]: Addison-Wesley Professional, 2010.
- CLEMENTS, P.; NORTHROP, L. *Software Product Lines: Practices and Patterns*. [S.l.]: Addison-Wesley, 2001.

- CORDEIRO, A. F.; OLIVEIRAJR, E. Size, Coupling and Cohesion Metrics for Product-Line Architecture Evaluation: Proposal and Experimental Validation. *Journal of Computer Science*, v. 14, n. 3, p. 408–422, 2018.
- CZARNECKI, K.; EISENECKER, U. W. *Generative Programming: Methods, Tools, and Applications*. [S.l.]: ACM Press/Addison-Wesley Publishing Co., 2000.
- DANIEL, W. *Applied nonparametric statistics*. [S.l.]: PWS-Kent Publ., 1990. (The Duxbury advanced series in statistics and decision sciences).
- DEELSTRA, S.; SINNEMA, M.; BOSCH, J. Variability assessment in software product families. *Inf. Softw. Technol.*, Butterworth-Heinemann, v. 51, n. 1, p. 195–218, 2009.
- DERMEVAL, D.; VILELA, J.; BITTENCOURT, I. I.; CASTRO, J.; ISOTANI, S.; BRITO, P. A Systematic Review on the Use of Ontologies in Requirements Engineering. In: *2014 Brazilian Symposium on Software Engineering (SBES)*. [S.l.: s.n.], 2014. p. 1–10.
- DEURSEN, A. van; HOFMEISTER, C.; KOSCHKE, R.; MOONEN, L.; RIVA, C. Symphony: view-driven software architecture reconstruction. In: *Fourth Working IEEE/IFIP Conference on Software Architecture*. [S.l.: s.n.], 2004. p. 122–132.
- DHUNGANA, D.; GRÜNBACHER, P.; RABISER, R. The dopler meta-tool for decision-oriented variability modeling: A multiple case study. *Automated Software Engg.*, Kluwer Academic Publishers, v. 18, n. 1, p. 77–114, 2011.
- DING, W.; LIANG, P.; TANG, A.; VLIET, H. van. Knowledge-based approaches in software documentation: A systematic literature review. *Information and Software Technology*, v. 56, n. 6, p. 545 – 567, 2014.
- DUCASSE, S.; POLLET, D. Software Architecture Reconstruction: A Process-Oriented Taxonomy. *IEEE Transactions on Software Engineering*, v. 35, n. 4, p. 573–591, 2009.
- DYBÅ, T.; DINGSØYR, T. Empirical studies of agile software development: A systematic review. *Inf. Softw. Technol.*, Butterworth-Heinemann, v. 50, n. 9-10, p. 833–859, 2008.
- EASTERBROOK, S. Empirical research methods for software engineering. In: *Proceedings of the Twenty-second IEEE/ACM International Conference on Automated Software Engineering*. [S.l.]: ACM, 2007.
- EIXELSBERGER, W. *Software Architecture Recovery of Product Lines*. Phd Thesis (PhD Thesis) — Universität Klagenfurt Fakultät für Wirtschaftswissenschaften und Informatik, 2000.
- EIXELSBERGER, W.; KALAN, M.; OGRIS, M.; BECKMAN, H.; BELLAY, B.; GALL, H. Recovery of architectural structure: A case study. In: *Development and Evolution of Software Architectures for Product Families*. [S.l.]: Springer Berlin Heidelberg, 1998, (Lecture Notes in Computer Science, v. 1429). p. 89–96.

- EIXELSBERGER, W.; WARHOLM, L.; KLÖSCH, R.; GALL, H. Software architecture recovery of embedded software. In: *Proceedings of the 19th International Conference on Software Engineering*. [S.l.]: ACM, 1997.
- EKLUND, U.; JONSSON, N.; BOSCH, J.; ERIKSSON, A. A reference architecture template for software-intensive embedded systems. In: *Proceedings of the WICSA/ECISA 2012 Companion Volume*. [S.l.]: ACM, 2012. p. 104–111.
- EPPINGER, S. D.; BROWNING, T. R. *Design Structure Matrix Methods and Applications*. [S.l.]: MIT Press, 2012.
- FENTON, N.; BIEMAN, J. *Software Metrics: A Rigorous and Practical Approach, Third Edition*. 3rd. ed. [S.l.]: CRC Press, Inc., 2014. ISBN 1439838224, 9781439838228.
- FIGUEIREDO, E.; CACHO, N.; SANT’ANNA, C.; MONTEIRO, M.; KULESZA, U.; GARCIA, A.; SOARES, S.; FERRARI, F.; KHAN, S.; FILHO, F. C.; DANTAS, F. Evolving software product lines with aspects: An empirical study on design stability. In: *Proceedings of the 30th International Conference on Software Engineering*. [S.l.]: ACM, 2008. p. 261–270.
- FISCHER, S.; LINSBAUER, L.; LOPEZ-HERREJON, R. E.; EGYED, A. Enhancing clone-and-own with systematic reuse for developing software variants. In: *2014 IEEE International Conference on Software Maintenance and Evolution*. [S.l.: s.n.], 2014. p. 391–400.
- FOWLER, M. *Refactoring: improving the design of existing code*. [S.l.]: Pearson Education India, 2009.
- FRITSCH, C.; LEHN, A.; STROHM, D. T.; GMBH, R. B. Evaluating variability implementation mechanisms. In: *International Workshop on Product Line Engineering The Early Steps: Planning, Modeling, and Managing*. [S.l.: s.n.], 2002. p. 59–64.
- GALL, H.; JAZAYERI, M.; KLÖSCH, R.; LUGMAYR, W.; TRAUSMUTH, G. Architecture recovery in ares. In: *Joint Proceedings of the Second International Software Architecture Workshop (ISAW-2) and International Workshop on Multiple Perspectives in Software Development (Viewpoints ’96) on SIGSOFT ’96 Workshops*. [S.l.]: ACM, 1996. p. 111–115.
- GALSTER, M.; AVGERIOU, P. Handling variability in software architecture: Problems and implications. In: *Proceedings of the 2011 Ninth Working IEEE/IFIP Conference on Software Architecture*. [S.l.]: IEEE Computer Society, 2011. p. 171–180.
- GALSTER, M.; AVGERIOU, P. The notion of variability in software architecture: Results from a preliminary exploratory study. In: *Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems*. [S.l.]: ACM, 2011. p. 59–67.

- GALSTER, M.; WEYNS, D.; AVGERIOU, P.; BECKER, M. Variability in software architecture: views and beyond. *SIGSOFT Softw. Eng. Notes*, ACM, v. 37, n. 6, p. 1–9, 2013.
- GARCIA, J.; IVKOVIC, I.; MEDVIDOVIC, N. A comparative analysis of software architecture recovery techniques. In: *2013 IEEE/ACM 28th International Conference on Automated Software Engineering*. [S.l.: s.n.], 2013. p. 486–496.
- GARCIA, J.; KRKA, I.; MATTMANN, C.; MEDVIDOVIC, N. Obtaining ground-truth software architectures. In: *International Conference on Software Engineering*. [S.l.]: IEEE Press, 2013. p. 901–910.
- GARCIA, J.; KRKA, I.; MEDVIDOVIC, N.; DOUGLAS, C. A framework for obtaining the ground-truth in architectural recovery. In: *Proceedings of the 2012 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture*. [S.l.]: IEEE Computer Society, 2012. p. 292–296.
- GARLAN, D. Software architecture: A roadmap. In: *Proceedings of the Conference on The Future of Software Engineering*. [S.l.]: ACM, 2000. p. 91–101.
- GOMAA, H. *Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures*. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 2004.
- GREENWOOD, P.; BARTOLOMEI, T.; FIGUEIREDO, E.; DOSEA, M.; GARCIA, A.; CACHO, N.; SANT’ANNA, C.; SOARES, S.; BORBA, P.; KULESZA, U.; RASHID, A. On the impact of aspectual decompositions on design stability: An empirical study. In: _____. *21st European Conference on Object-Oriented Programming*. [S.l.]: Springer Berlin Heidelberg, 2007. p. 176–200.
- HAIDER, U.; WOODS, E.; BASHROUSH, R. Representing variability in software architecture: A systematic literature review. *International Journal of Software Engineering and Computer Systems*, Universiti Malaysia Pahang, v. 4, n. 2, p. 19–37, August 2018.
- HARRIS, D. R.; REUBENSTEIN, H. B.; YEH, A. S. Reverse engineering to the architectural level. In: *17th International Conference on Software Engineering*. [S.l.]: ACM, 1995. p. 186–195.
- HENARD, C.; PAPADAKIS, M.; PERROUIN, G.; KLEIN, J.; HEYMANS, P.; TRAON, Y. L. Bypassing the combinatorial explosion: Using similarity to generate and prioritize t-wise test configurations for software product lines. *IEEE Transactions on Software Engineering*, v. 40, n. 7, p. 650–670, 2014.
- HESSE-BIBER, S. *Mixed Methods Research: Merging Theory with Practice*. [S.l.]: Guilford Publications, 2010.
- HILLIARD, R. On representing variation. In: *1st International Workshop on Variability in Software Product Line Architectures*. [S.l.]: ACM, 2010.

ISO/IEC/IEEE Systems and software engineering – Architecture description. *ISO/IEC/IEEE 42010:2011(E) (Revision of ISO/IEC 42010:2007 and IEEE Std 1471-2000)*, p. 1–46, Dec 2011.

JARING, M.; BOSCH, J. Expressing product diversification — categorizing and classifying variability in software product family engineering. *International Journal of Software Engineering and Knowledge Engineering*, v. 14, n. 05, p. 449–470, 2004.

KANG, K.; COHEN, S.; HESS, J.; NOVAK, W.; PETERSON, A. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. Pittsburgh, PA, 1990.

KäSTNER, C.; APEL, S.; KUHLEMANN, M. Granularity in software product lines. In: *Proceedings of the 30th International Conference on Software Engineering*. [S.l.]: ACM, 2008. p. 311–320.

KäSTNER, C.; APEL, S.; RAHMAN, S. S. ur; ROSENMÜLLER, M.; BATORY, D.; SAAKE, G. On the impact of the optional feature problem: Analysis and case studies. In: *Proceedings of the 13th International Software Product Line Conference*. [S.l.]: Carnegie Mellon University, 2009. (SPLC '09), p. 181–190.

KAZMAN, R.; CARRIÈRE, S. J. Playing Detective: Reconstructing Software Architecture from Available Evidence. *Automated Software Engineering*, v. 6, n. 2, p. 107–138, 1999.

KIM, S. D.; HER, J. S.; CHANG, S. H. A theoretical foundation of variability in component-based development. *Information and Software Technology*, v. 47, n. 10, p. 663 – 673, 2005.

KITCHENHAM, B.; CHARTERS, S. Guidelines for performing Systematic Literature Reviews in Software Engineering. *Software Engineering Group School of*, ACM Press, v. 2, p. 1051, 2007.

KOSCHKE, R.; FRENZEL, P.; BREU, A.; ANGSTMANN, K. Extending the reflexion method for consolidating software variants into product lines. *Software Quality Journal*, Springer US, v. 17, n. 4, p. 331–366, 2009.

KRUCHTEN, P. The 4+1 view model of architecture. *IEEE Softw.*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 12, n. 6, p. 42–50, 1995. ISSN 0740-7459.

KRUEGER, C. W. Applied software product line engineering. In: _____. [S.l.]: Auerbach Publications, 2009. chap. New Methods behind a New Generation of Software product line Successes, p. 39–60.

KRÜGER, J.; FENSKE, W.; THÜM, T.; APORIUS, D.; SAAKE, G.; LEICH, T. 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*. [S.l.]: ACM, 2018. (SPLC '18).

- LETHBRIDGE, T. C.; SIM, S. E.; SINGER, J. Studying software engineers: Data collection techniques for software field studies. *Empirical Software Engineering*, v. 10, n. 3, p. 311–341, 2005.
- LIMA, C. Product line architecture recovery: An approach proposal (extended abstract). In: *39th International Conference on Software Engineering Companion - Doctoral Symposium*. [S.l.]: ACM, 2017.
- LIMA, C.; ASSUNCAO, W. K. G.; MARTINEZ, J.; MACHADO, I. do C.; CHAVEZ, C. von F. G.; MENDONCA, W. D. F. Towards an automated product line architecture recovery: The apo-games case study. In: *VII Brazilian Symposium on Software Components, Architectures, and Reuse*. [S.l.]: ACM, 2018. (SBCARS '18), p. 33–42.
- LIMA, C.; CHAVEZ, C. A systematic review on metamodels to support product line architecture design. In: *Proceedings of the 30th Brazilian Symposium on Software Engineering*. [S.l.]: ACM, 2016. p. 13–22.
- LIMA, C.; CHAVEZ, C.; ALMEIDA, E. S. de. Investigating the recovery of product line architectures: An approach proposal. In: _____. *Mastering Scale and Complexity in Software Reuse: 16th International Conference on Software Reuse*. Springer International Publishing, 2017. p. 201–207. ISBN 978-3-319-56856-0. Available at: <http://dx.doi.org/10.1007/978-3-319-56856-0_15>.
- LIMA, C.; MACHADO, I. do C.; ALMEIDA, E. S. de; CHAVEZ, C. von F. G. Recovering the Product Line Architecture of the Apo-Games. In: *Proceedings of the 22nd International Systems and Software Product Line Conference*. [S.l.]: ACM, 2018. (SPLC '18).
- LIMA-NETO, C. R.; CARDOSO, M. P. S.; CHAVEZ, C. v. F. G.; ALMEIDA, E. S. d. Initial evidence for understanding the relationship between product line architecture and software architecture recovery. In: *2015 IX Brazilian Symposium on Components, Architectures and Reuse Software (SBCARS)*. [S.l.: s.n.], 2015. p. 40–49.
- LIMA-NETO, C. R.; CHAVEZ, C. von F. G.; ALMEIDA, E. S. de. Characterizing Product Line Architecture Recovery. *ICSE 2017 PhD and Young Researchers Warm Up Symposium, Co-located with CBSOft 2014*, 2014.
- LIMA-NETO, C. R.; CHAVEZ, C. von F. G.; ALMEIDA, E. S. de; ROST, D.; NAAB, M. Unveiling architecture documentation: Brazilian stakeholders in perspective. *CoRR*, TR-PGCOMP-002/2015, 2015. Available at: <<http://arxiv.org/abs/1510.06229>>.
- LINDEN, F. J. van der; SCHMID, K.; ROMMES, E. *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*. [S.l.]: Springer, 2007.
- LINSBAUER, L.; LOPEZ-HERREJON, R. E.; EGYED, A. Variability extraction and modeling for product variants. *Software & Systems Modeling*, Springer, p. 1–21, 2016.

- LOSAVIO, F.; ORDAZ, O.; LÉVY, N.; BAIOTTO, A. Graph modelling of a refactoring process for product line architecture design. In: *XXXIX Latin American Computing Conference*. [S.l.: s.n.], 2013. p. 1–12.
- MAHDAVI-HEZAVEHI, S.; GALSTER, M.; AVGERIOU, P. Variability in quality attributes of service-based software systems: A systematic literature review. *Inf. Softw. Technol.*, Butterworth-Heinemann, v. 55, n. 2, p. 320–343, 2013.
- MANCORIDIS, S.; MITCHELL, B.; CHEN, Y.; GANSNER, E. Bunch: a clustering tool for the recovery and maintenance of software system structures. In: *IEEE International Conference on Software Maintenance Proceedings*. [S.l.: s.n.], 1999. p. 50–59.
- MANCORIDIS, S.; MITCHELL, B. S.; RORRES, C.; CHEN, Y.; GANSNER, E. R. Using automatic clustering to produce high-level system organizations of source code. In: *6th International Workshop on Program Comprehension*. [S.l.: s.n.], 1998. p. 45–52.
- MARTÍNEZ-FERNÁNDEZ, S.; AYALA, C. P.; FRANCH, X.; MARQUES, H. M. Safe and secure software reuse: 13th international conference on software reuse, icsr 2013, pisa, june 18-20. proceedings. In: _____. [S.l.]: Springer Berlin Heidelberg, 2013. chap. REARM: A Reuse-Based Economic Model for Software Reference Architectures, p. 97–112.
- MEDVIDOVIC, N.; EGYED, A.; GRÜNBACHER, P. Stemming Architectural Erosion by Coupling Architectural Discovery and Recovery. In: *2nd Int'l Software Requirements to Architectures Workshop*. [S.l.: s.n.], 2003. p. 61–68.
- MEINICKE, J.; THÜM, T.; SCHRÖTER, R.; KRIETER, S.; BENDUHN, F.; SAAKE, G.; LEICH, T. Featureide: Taming the preprocessor wilderness. In: *Proceedings of the 38th International Conference on Software Engineering Companion*. [S.l.]: ACM, 2016. p. 629–632.
- MENDONCA, N. C.; KRAMER, J. Requirements for an effective architecture recovery framework. In: *Joint Proceedings of the Second International Software Architecture Workshop (ISAW-2) and International Workshop on Multiple Perspectives in Software Development (Viewpoints '96) on SIGSOFT '96 Workshops*. [S.l.]: ACM, 1996. (ISAW '96), p. 101–105.
- MENDONCA, N. C.; KRAMER, J. Developing an approach for the recovery of distributed software architectures. In: *6th International Workshop on Program Comprehension*. [S.l.: s.n.], 1998. p. 28–36.
- MOHAN, K.; RAMESH, B. Tracing variations in software product families. *Commun. ACM*, ACM, v. 50, n. 12, p. 68–73, 2007.
- MOLLAH, M.; ISLAM, K.; ISLAM, S. Next generation of computing through cloud computing technology. In: *2012 25th IEEE Canadian Conference on Electrical Computer Engineering*. [S.l.: s.n.], 2012. p. 1–6.

- MOON, M.; CHAE, H.; YEOM, K. A Metamodel Approach to Architecture Variability in a Product Line. In: *Reuse of Off-the-Shelf Components. ICSR 2006. Lecture Notes in Computer Science*. [S.l.]: Springer, Berlin, Heidelberg, 2006. v. 4039.
- NAKAGAWA, E. Y.; ANTONINO, P. O.; BECKER, M. 5th european conference on software architecture. In: _____. [S.l.]: Springer Berlin Heidelberg, 2011. chap. Reference Architecture and Product Line Architecture: A Subtle But Critical Difference, p. 207–211.
- NATRELLA, M. *NIST/SEMATECH e-Handbook of Statistical Methods*. NIST/SEMATECH, 2010. Available at: <<http://www.itl.nist.gov/div898/handbook/>>.
- NORTHROP, L. M. SEI's Software Product Line Tenets. *IEEE Software*, IEEE Computer Society, v. 19, n. 4, p. 32–40, 2002.
- OLIVEIRA-JUNIOR, E. A.; GIMENES, I.; MALDONADO, J. A metric suite to support software product line architecture evaluation. In: *XXXIV Conferencia Latinoamericana de Informatica*. [S.l.: s.n.], 2008. p. 489–498.
- PASHOV, I.; RIEBISCH, M. Using feature modeling for program comprehension and software architecture recovery. In: *Engineering of Computer-Based Systems, 2004. Proceedings. 11th IEEE International Conference and Workshop on the*. [S.l.: s.n.], 2004. p. 406–417.
- PETERSEN, W. A set-theoretical approach for the induction of inheritance hierarchies. *Electronic Notes in Theoretical Computer Science*, v. 53, p. 296 – 308, 2004.
- PINZGER, M.; GALL, H.; GIRARD, J.-F.; KNODEL, J.; RIVA, C.; PASMAN, W.; BROERSE, C.; WIJNSTRA, J. Architecture recovery for product families. In: _____. *Software Product-Family Engineering*. [S.l.]: Springer Berlin Heidelberg, 2003. (Lecture Notes in Computer Science, v. 3014), chap. 26, p. 332–351.
- POHL, K.; BÖCKLE, G.; LINDEN, F. van der. *Software Product Line Engineering: Foundations, Principles, and Techniques*. [S.l.]: Springer-Verlag New York, Inc., 2005. 467 p.
- POLLET, D.; DUCASSE, S.; POYET, L.; ALLOUI, I.; CIMPAN, S.; VERJUS, H. Towards a process-oriented software architecture reconstruction taxonomy. In: *11th European Conference on Software Maintenance and Reengineering*. [S.l.: s.n.], 2007. p. 137–148.
- ROST, D.; NAAB, M.; LIMA, C.; CHAVEZ, C. von F. G. Software architecture documentation for developers: A survey. In: *Proceedings of the 7th European Conference on Software Architecture*. [S.l.]: Springer-Verlag, 2013. p. 72–88.
- RUBIN, J.; CHECHIK, M. Locating distinguishing features using diff sets. In: *ACM. 27th IEEE/ACM International Conference on Automated Software Engineering*. [S.l.], 2012. p. 242–245.

- SCHMID, K.; JOHN, I. A customizable approach to full lifecycle variability management. *Sci. Comput. Program.*, Elsevier North-Holland, Inc., v. 53, n. 3, p. 259–284, 2004.
- SHATNAWI, A.; SERIAI, A.; SAHRAOUI, H. Recovering Architectural Variability of a Family of Product Variants. In: *Software Reuse for Dynamic Systems in the Cloud and Beyond*. [S.l.]: Springer International Publishing, 2015, (Lecture Notes in Computer Science, v. 8919). p. 17–33.
- SHATNAWI, A.; SERIAI, A.-D.; SAHRAOUI, H. Recovering software product line architecture of a family of object-oriented product variants. *Journal of Systems and Software*, 2016.
- SHAW, M.; GARLAN, D. *Software Architecture: Perspectives on an Emerging Discipline*. [S.l.]: Prentice-Hall, Inc., 1996.
- SHULL, F.; BABAR, M. A.; CHEN, L. Managing variability in software product lines. *IEEE Software*, IEEE Computer Society, v. 27, p. 89–91, 94, 2010.
- SINKALA, Z. T.; BLOM, M.; HEROLD, S. A mapping study of software architecture recovery for software product lines. In: *Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings*. [S.l.]: ACM, 2018. (ECSA '18), p. 49:1–49:7.
- SINNEMA, M.; DEELSTRA, S. Classifying variability modeling techniques. *Inf. Softw. Technol.*, Butterworth-Heinemann, v. 49, n. 7, p. 717–739, 2007.
- SOLINGEN, R. van; BASILI, V.; CALDIERA, G.; ROMBACH, H. D. *Goal Question Metric (GQM) Approach*. [S.l.]: John Wiley Sons, Inc., 2002.
- SOUZA-FILHO, E. D.; CAVALCANTI, R. O.; NEIVA, D. F.; OLIVEIRA, T. H.; LISBOA, L. B.; ALMEIDA, E. S.; MEIRA, S. R. L. Evaluating Domain Design Approaches Using Systematic Review. In: *Proceedings of the 2nd European Conference on Software Architecture*. [S.l.]: Springer-Verlag, 2008. (ECSA '08), p. 50–65.
- STAVROPOULOU, I.; GRIGORIOU, M.; KONTOGIANNIS, K. Case study on which relations to use for clustering-based software architecture recovery. *Empirical Software Engineering*, v. 22, n. 4, p. 1717–1762, Aug 2017.
- SVAHNBERG, M.; GURP, J. van; BOSCH, J. A taxonomy of variability realization techniques: Research articles. *Softw. Pract. Exper.*, John Wiley & Sons, Inc., v. 35, n. 8, p. 705–754, 2005.
- TAYLOR, R.; MEDVIDOVIC, N.; DASHOFY, E. *Software Architecture: Foundations, Theory, and Practice*. [S.l.]: Wiley, 2009.
- THIEL, S.; HEIN, A. Modeling and using product line variability in automotive systems. *IEEE Softw.*, IEEE Computer Society Press, v. 19, n. 4, p. 66–72, 2002.

- THIEL, S.; HEIN, A. Systematic Integration of Variability into Product Line Architecture Design. In: *Software Product Lines, Second Int. Conf. , SPLC 2, San Diego, CA, USA, August 19-22, 2002, Proc.* [S.l.]: Springer, 2002. (Lecture Notes in Computer Science, v. 2379), p. 130–153.
- TILLEY, S. R.; PAUL, S.; SMITH, D. B. Towards a framework for program understanding. In: *Fourth Workshop on Program Comprehension.* [S.l.: s.n.], 1996. p. 19–28.
- TRUJILLO, S.; BATORY, D.; DIAZ, O. Feature oriented model driven development: A case study for portlets. In: *Proceedings of the 29th International Conference on Software Engineering.* [S.l.]: IEEE Computer Society, 2007. p. 44–53.
- VERLAGE, M.; KIESGEN, T. Five years of product line engineering in a small company. In: *Proceedings of the 27th International Conference on Software Engineering.* [S.l.]: ACM, 2005. p. 534–543.
- WATERS, R. G.; CHIKOFFSKY, E. Reverse engineering: Progress along many dimensions. *Commun. ACM*, ACM, v. 37, n. 5, p. 22–25, 1994.
- WEISS, D. M.; LAI, C. T. R. *Software Product-line Engineering: A Family-based Software Development Process.* [S.l.]: Addison-Wesley Longman Publishing Co., Inc., 1999.
- WIERINGA, R.; MAIDEN, N.; MEAD, N.; ROLLAND, C. Requirements engineering paper classification and evaluation criteria: a proposal and a discussion. *Requir. Eng.*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, v. 11, n. 1, p. 102–107, 2005.
- WILLE, D.; BABUR Önder; CLEOPHAS, L.; SEIDL, C.; BRAND, M. van den; SCHAEFER, I. Improving custom-tailored variability mining using outlier and cluster detection. *Science of Computer Programming*, v. 163, p. 62 – 84, 2018. ISSN 0167-6423.
- WOHLIN, C.; RUNESON, P.; HÖST, M.; OHLSSON, M. C.; REGNELL, B. *Experimentation in Software Engineering.* [S.l.]: Springer, 2012.
- WU, Y.; YANG, Y.; PENG, X.; QIU, C.; ZHAO, W. Recovering object-oriented framework for software product line reengineering. In: *Proceedings of the 12th International Conference on Top Productivity Through Software Reuse.* [S.l.]: Springer-Verlag, 2011. p. 119–134.
- YOUNG, T. J. *Using AspectJ to Build a Software Product Line for Mobile Devices.* Master's Thesis (Master's Thesis) — The University of British Columbia (UBC), 2005.
- ZHANG, S.; ZHANG, S.; CHEN, X.; HUO, X. Cloud Computing Research and Development Trend. In: *Second International Conference on Future Networks.* [S.l.: s.n.], 2010. p. 93–97.
- ZHANG, T.; DENG, L.; WU, J.; ZHOU, Q.; MA, C. Some metrics for accessing quality of product line architecture. In: IEEE. *International Conference on Computer Science and Software Engineering.* [S.l.], 2008. v. 2, p. 500–503.

MAPPING STUDY ON PLA RECOVERY - PRIMARY STUDIES AND DATA SET

This appendix lists the primary studies analyzed in the survey on the relationship between SAR and PLA, earlier addressed in Chapter 3. We also lists the data set used in the study and the included studies per venue.

A.1 LIST OF JOURNALS

Table A.1 List of Journals

Journals
ACM Computing Survey
ACM Transactions on Software Engineering and Methodology (TOSEM)
Annals of Software Engineering
Automated Software Engineering
ELSEVIER Information and Software Technology (IST)
ELSEVIER Journal of Systems and Software (JSS)
IEEE Software
IEEE Transactions on Software Engineering
Journal of Systems and Software
Software Process: Improvement and Practice
Software Practice and Experience
Journal of Software Maintenance Research and Practice
Journal of Systems and Software (JSS)
Software Practice and Experience (SPE) Journal
Software Quality Journal
Software Testing, Verification and Reliability

A.2 LIST OF CONFERENCES

Table A.2 List of Conferences

Acronym	Conference Name
APSEC	Asia Pacific Software Engineering Conference
ASE	International Conference on Automated Software Engineering
CAiSE	International Conference on Advanced Information Systems Engineering
CBSE	International Symposium on Component-based Software Engineering
COMPSAC	International Computer Software and Applications Conference
ECBS	International Conference and Workshop on the Engineering of Computer Based Systems
ECOWS	European Conference on Web Services
ECSA	European Conference on Software Architecture
ESEC	European Software Engineering Conference
ESEM	Empirical Software Engineering and Measurement
FASE	Fundamental Approaches to Software Engineering
ICCBSS	International Conference on Composition-Based Software Systems
ICSE	International Conference on Software Engineering
ICSM	International Conference on Software Maintenance
ICSR	International Conference on Software Reuse
ICST	International Conference on Software Testing, Verification and Validation
ICWS	International Conference on Web Services
ISSRE	International Symposium on Software Reliability Engineering
GPCE	International Conference on Generative Programming and Component Engineering
MODEL	International Conference on Model Driven Engineering Languages and Systems
MoTiP	Workshop on Model-based Testing in Practice
OOPSLA	ACM SIGPLAN conference on Object-Oriented Programming, Systems, Languages, and App.
PROFES	International Conference on Product Focused Software Development and Process Improvement
QoSA	International Conference on the Quality of Software Architectures
QSIC	International Conference on Quality Software
ROSATEA	International Workshop on The Role of Software Architecture in Testing and Analysis
SAC	Annual ACM Symposium on Applied Computing
SEAA	Euromicro Conference on Software Engineering and Advanced Applications
SEKE	International Conference on Software Engineering and Knowledge Engineering
SPLC	Software Product Line Conference
VaMoS	Variability Modelling of Software-Intensive Systems
WICSA	Working IEEE/IFIP Conference on Software Architecture

A.3 PRIMARY STUDIES

Table A.3 Selected primary studies

ID	Title	Author(s)	Venue
S1	Software Architecture Recovery of a Program Family	W. Eixelsberger	ICSE'98
S2	Software Architecture Recovery of Product Lines	W. Eixelsberger	Ph.D. Th.'00
S3	MAP - Mining Architectures for Product Line Evaluations	C. Stoermer	WICSA'01
S4	Architecture Reconstruction Guidelines	R. Kazman	Tec. Rep.'03
S5	Architecture Recovery for Product Families	M. Pinzger	PFE'03
S6	Using Feature Modeling for Program Comprehension and Software Architecture Recovery	I. Pashov	ECBS'04
S7	View-based Software Architecture Reconstruction	C. Riva	Ph.D. Th.'04
S8	Developing Tools for Reverse Engineering in a Software Product-Line Architecture	C. Chiang	IRI'04
S9	Feature-Oriented Re-engineering of Legacy Systems into Product Line Assets - a Case Study	K. C. Kang	SPLC'05
S10	Static Evaluation of Software Architectures	J. Knodel	CSMR'06
S11	Towards a Process-Oriented Software Architecture Reconstruction Taxonomy	D. Pollet	CSMR'07
S12	Architectural elements recovery and quality evaluation to assist in reference architectures specification	A. Vasconcelos	SEKE'07
S13	Architecture Recovery and Evaluation Aiming at Program Understanding and Reuse	A. Vasconcelos	QoSA'07
S14	Extending the Reflexion Method for Consolidating Software Variants into Product Lines	P. Frenzel	WCRE'07
S15	Extending the reflexion method for consolidating software variants into product lines	R. Koschke	Soft Qual J'09
S16	Software Architecture Reconstruction: A Process Oriented Taxonomy	S. Ducasse	IEEE Tr.'09
S17	Evaluating reuse and program understanding in ArchMine architecture recovery approach	A. Vasconcelos	JIS'10
S18	Variability Management for Software Product-Line Architecture Development	Y. Kim	JSEKE'11
S19	Recovering object-oriented framework for software product line reengineering	Y. Wu	ICSR'11

Continued on next page...

ID	Title	Author(s)	Venue
S20	Exploring the Use of Reference Architectures in the Development of Product Line Artifacts	E. Nakagawa	SPLC'11
S21	Reverse Engineering Architectural Feature Models	M. Acher	ECSA'11
S22	Automatic Software Architecture Recovery: A Machine Learning Approach	H. Sajnani	ICPC'12
S23	History-sensitive heuristics for recovery of features in code of evolving program families	Nunes et al.	SPLC'12
S24	Towards a Process to design product line architectures based on reference architectures	Nakagawa et al.	SPLC'13
S25	A Comparative Analysis of Software Architecture Recovery Techniques	Garcia et al.	ASE'13
S26	Graph Modelling of a Refactoring Process for Product Line Architecture Design	F. Losavio	CLEI'13
S27	Architectural Bad Smells in Software Product Lines: An Exploratory Study	Sica et al.	WICSA'14
S28	ArchViz: a Tool to Support Architecture Recovery Research	Zapalowski et al.	CBSof't'14
S29	Recovering Architectural Variability of a Family of Product Variants	Shatnawi et al.	ICSR'15
S30	Comparing Software Architecture Recovery Techniques Using Accurate Dependencies	Thibaud et al.	ICSE'15
S31	Variability extraction and modeling for product variants	Linsbauer et al.	sof.sys.Model'16
S32	Exploring the combination of software visualization and data clustering in the software architecture recovery process	R. Paiva	SAC'16
S33	System Architecture Recovery based on Software Structure Model	A. Darvas	WICSA'16
S34	Recovering software product line architecture of a family of object-oriented product variants	Shatnawi et al.	JSS'16
S35	Discovering Software Architectures with Search-based Merge of UML Model Variants	W. Assunção	ICSR'17

RECOVERING PLA FROM VARIANTS' SOURCE CODE – ADDITIONAL MATERIAL

This appendix describes the additional material for Chapter 4.

B.1 MOTIVATING EXAMPLE ADDITIONAL MATERIAL

B.1.1 MobileMedia Variants' extracted architecture

Figure B.1 presents the extracted information from MobileMedia V4.

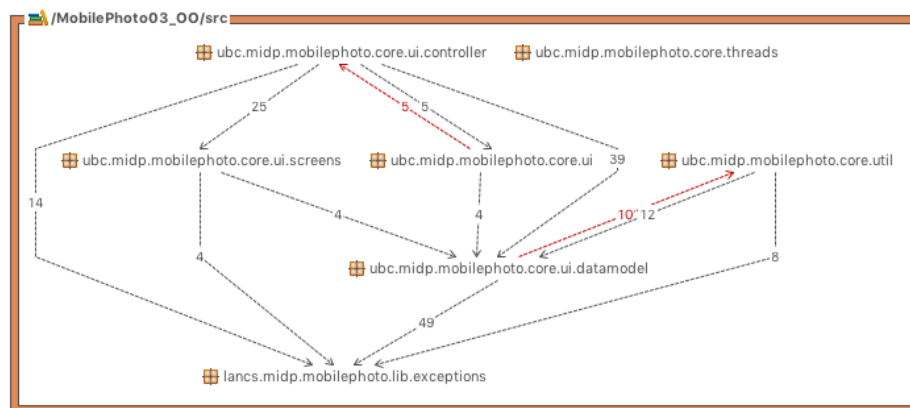


Figure B.1 Extracted information from MobileMedia Variant 4 (packages and relations)

Figure B.2 presents the extracted information from MobileMedia V5.

Figure B.3 presents the extracted information from MobileMedia V6.

Figure B.4 presents the extracted information from MobileMedia V7.

Figure B.5 presents the extracted information from MobileMedia V8.

B.1.2 MobileMedia SPL recovered PLA

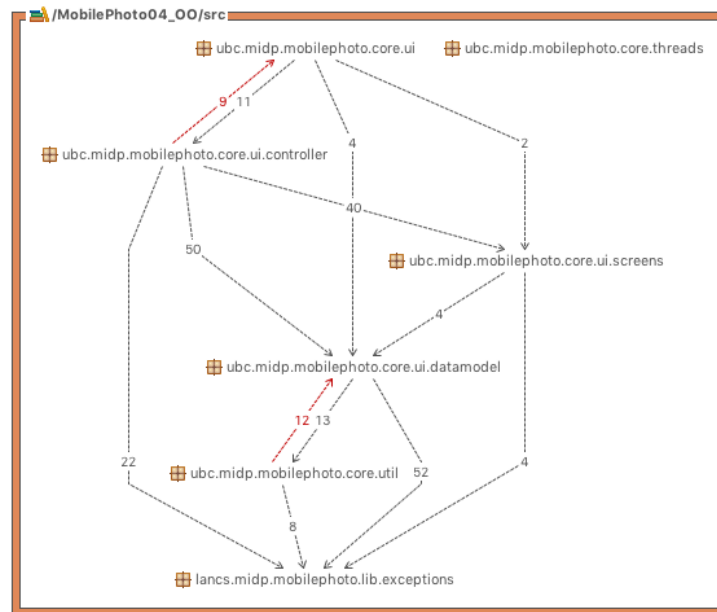


Figure B.2 Extracted information from MobileMedia Variant 5 (packages and relations)

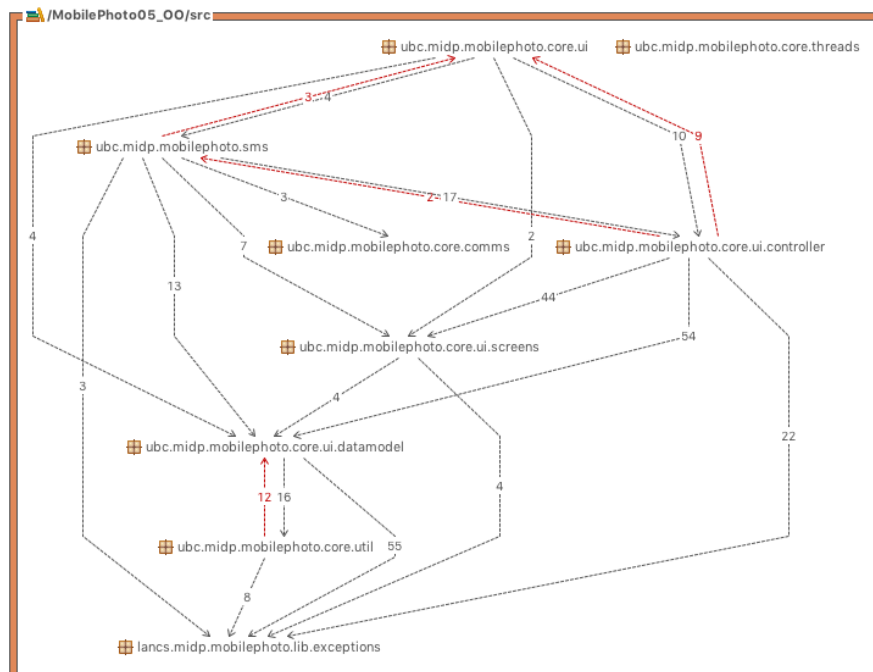


Figure B.3 Extracted information from MobileMedia Variant 6 (packages and relations)

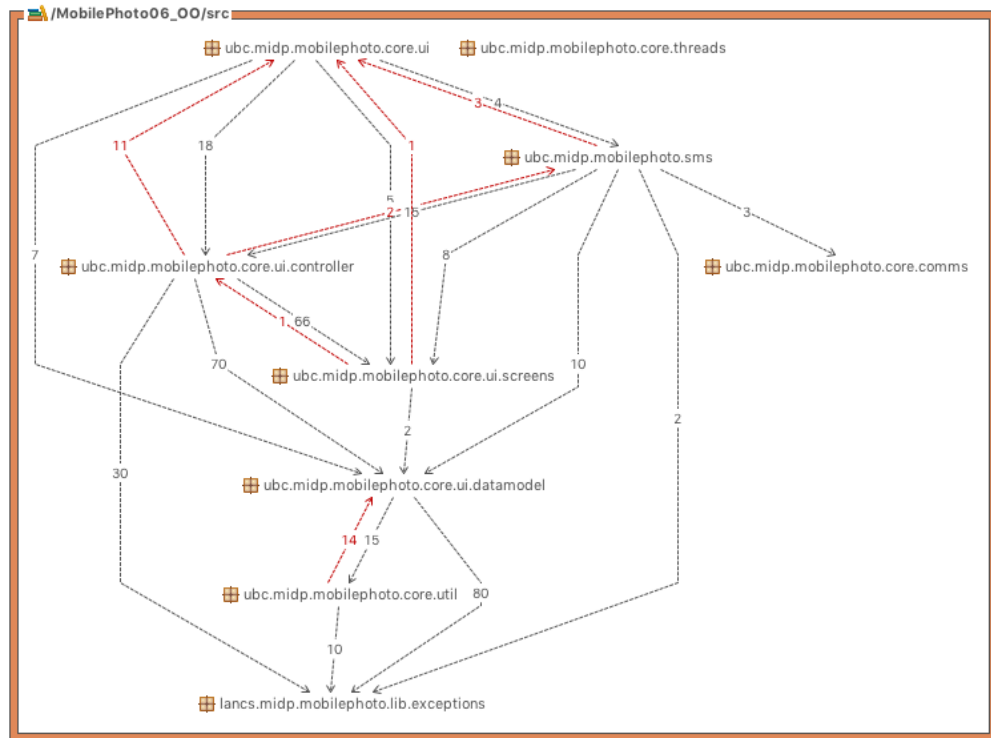


Figure B.4 Extracted information from MobileMedia Variant 7 (packages and relations)

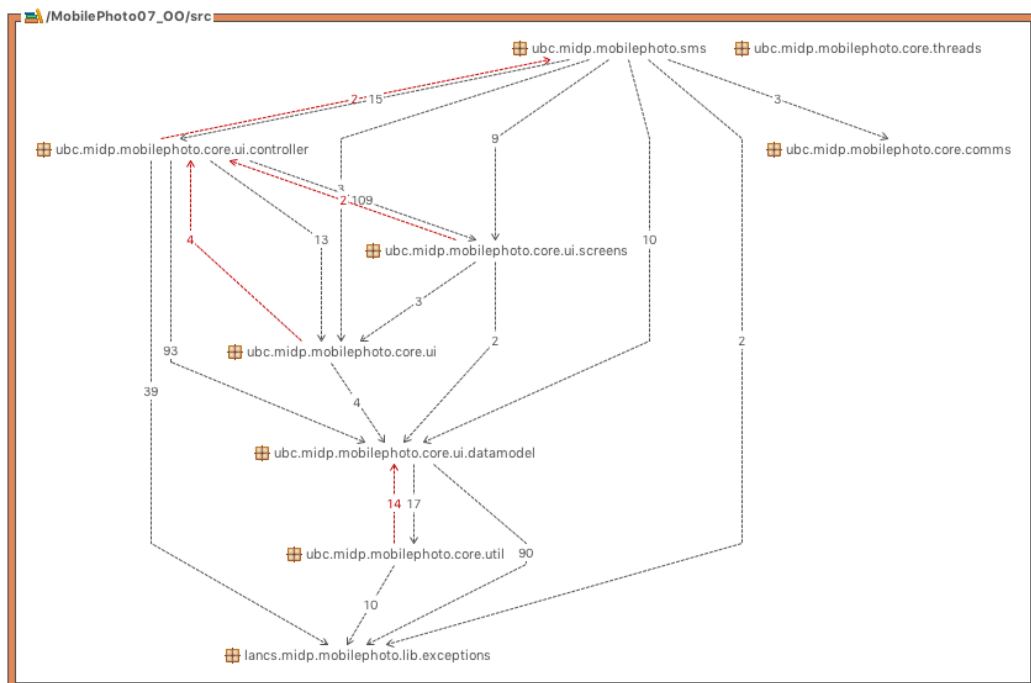


Figure B.5 Extracted information from MobileMedia Variant 8 (packages and relations)

```

COMPONENT METRICS

SSC:0.6666667

SVC:0.33333334

COMMON COMPONENTS TOTAL:6

VARIABILITY COMPONENTS TOTAL:3

RELATION METRICS

RSSC:0.3478261

RSVC:0.65217394

COMMON RELATIONS TOTAL:8

VARIABILITY RELATIONS TOTAL:15

```

Listing B.1 MobileMedia Metrics

```

Nodes:

lancs.midp.mobilephoto.lib.exceptions - Variant_2 Variant_3 Variant_4 Variant_5 Variant_6 Variant_7 Variant_8 - appear
(s) in 87% of the projects

ubc.midp.mobilephoto.core.comms - Variant_6 Variant_7 Variant_8 - appear(s) in 37% of the projects

ubc.midp.mobilephoto.core.threads - Variant_1 Variant_2 Variant_3 Variant_4 Variant_5 Variant_6 Variant_7 Variant_8 -
appear(s) in 100% of the projects

ubc.midp.mobilephoto.core.ui - Variant_1 Variant_2 Variant_3 Variant_4 Variant_5 Variant_6 Variant_7 Variant_8 -
appear(s) in 100% of the projects

ubc.midp.mobilephoto.core.ui.controller - Variant_1 Variant_2 Variant_3 Variant_4 Variant_5 Variant_6 Variant_7
Variant_8 - appear(s) in 100% of the projects

ubc.midp.mobilephoto.core.ui.datamodel - Variant_1 Variant_2 Variant_3 Variant_4 Variant_5 Variant_6 Variant_7
Variant_8 - appear(s) in 100% of the projects

ubc.midp.mobilephoto.core.ui.screens - Variant_1 Variant_2 Variant_3 Variant_4 Variant_5 Variant_6 Variant_7 Variant_8
- appear(s) in 100% of the projects

ubc.midp.mobilephoto.core.util - Variant_1 Variant_2 Variant_3 Variant_4 Variant_5 Variant_6 Variant_7 Variant_8 -
appear(s) in 100% of the projects

ubc.midp.mobilephoto.sms - Variant_6 Variant_7 Variant_8 - appear(s) in 37% of the projects

Dependencies:

ubc.midp.mobilephoto.core.ui ubc.midp.mobilephoto.core.ui.controller - Variant_1 Variant_2 Variant_3 Variant_4
Variant_5 Variant_6 Variant_7 Variant_8 - appear(s) in 100% of the projects

ubc.midp.mobilephoto.core.ui ubc.midp.mobilephoto.core.ui.datamodel - Variant_1 Variant_2 Variant_3 Variant_4
Variant_5 Variant_6 Variant_7 Variant_8 - appear(s) in 100% of the projects

ubc.midp.mobilephoto.core.ui ubc.midp.mobilephoto.core.ui.screens - Variant_5 Variant_6 Variant_7 - appear(s) in 37%
of the projects

ubc.midp.mobilephoto.core.ui ubc.midp.mobilephoto.sms - Variant_6 Variant_7 - appear(s) in 25% of the projects

ubc.midp.mobilephoto.core.ui.controller ubc.midp.mobilephoto.core.ui - Variant_1 Variant_2 Variant_3 Variant_4
Variant_5 Variant_6 Variant_7 Variant_8 - appear(s) in 100% of the projects

ubc.midp.mobilephoto.core.ui.controller ubc.midp.mobilephoto.core.ui.datamodel - Variant_1 Variant_2 Variant_3
Variant_4 Variant_5 Variant_6 Variant_7 Variant_8 - appear(s) in 100% of the projects

ubc.midp.mobilephoto.core.ui.controller ubc.midp.mobilephoto.core.ui.screens - Variant_1 Variant_2 Variant_3 Variant_4
Variant_5 Variant_6 Variant_7 Variant_8 - appear(s) in 100% of the projects

ubc.midp.mobilephoto.core.ui.controller lancs.midp.mobilephoto.lib.exceptions - Variant_2 Variant_3 Variant_4
Variant_5 Variant_6 Variant_7 Variant_8 - appear(s) in 87% of the projects

ubc.midp.mobilephoto.core.ui.controller ubc.midp.mobilephoto.sms - Variant_6 Variant_7 Variant_8 - appear(s) in 37%
of the projects

ubc.midp.mobilephoto.core.ui.datamodel ubc.midp.mobilephoto.core.util - Variant_1 Variant_2 Variant_3 Variant_4
Variant_5 Variant_6 Variant_7 Variant_8 - appear(s) in 100% of the projects

ubc.midp.mobilephoto.core.ui.datamodel lancs.midp.mobilephoto.lib.exceptions - Variant_2 Variant_3 Variant_4 Variant_5
Variant_6 Variant_7 Variant_8 - appear(s) in 87% of the projects

ubc.midp.mobilephoto.core.ui.screens ubc.midp.mobilephoto.core.ui.datamodel - Variant_1 Variant_2 Variant_3 Variant_4
Variant_5 Variant_6 Variant_7 Variant_8 - appear(s) in 100% of the projects

ubc.midp.mobilephoto.core.ui.screens lancs.midp.mobilephoto.lib.exceptions - Variant_2 Variant_3 Variant_4 Variant_5
Variant_6 - appear(s) in 62% of the projects

```

```

ubc.midp.mobilephoto.core.ui.screens ubc.midp.mobilephoto.core.ui - Variant_7 Variant_8 - appear(s) in 25% of the
  projects
ubc.midp.mobilephoto.core.ui.screens ubc.midp.mobilephoto.core.ui.controller - Variant_7 Variant_8 - appear(s) in 25%
  of the projects
ubc.midp.mobilephoto.core.util ubc.midp.mobilephoto.core.ui.datamodel - Variant_1 Variant_2 Variant_3 Variant_4
  Variant_5 Variant_6 Variant_7 Variant_8 - appear(s) in 100% of the projects
ubc.midp.mobilephoto.core.util lncs.midp.mobilephoto.lib.exceptions - Variant_2 Variant_3 Variant_4 Variant_5
  Variant_6 Variant_7 Variant_8 - appear(s) in 87% of the projects
ubc.midp.mobilephoto.sms lncs.midp.mobilephoto.lib.exceptions - Variant_6 Variant_7 Variant_8 - appear(s) in 37% of
  the projects
ubc.midp.mobilephoto.sms ubc.midp.mobilephoto.core.comms - Variant_6 Variant_7 Variant_8 - appear(s) in 37% of the
  projects
ubc.midp.mobilephoto.sms ubc.midp.mobilephoto.core.ui - Variant_6 Variant_7 Variant_8 - appear(s) in 37% of the
  projects
ubc.midp.mobilephoto.sms ubc.midp.mobilephoto.core.ui.controller - Variant_6 Variant_7 Variant_8 - appear(s) in 37% of
  the projects
ubc.midp.mobilephoto.sms ubc.midp.mobilephoto.core.ui.datamodel - Variant_6 Variant_7 Variant_8 - appear(s) in 37% of
  the projects
ubc.midp.mobilephoto.sms ubc.midp.mobilephoto.core.ui.screens - Variant_6 Variant_7 Variant_8 - appear(s) in 37% of
  the projects

```

Listing B.2 MobileMedia Report of the PLA recovery

EXPLORATORY STUDY ON PRODUCT LINE ARCHITECTURE RECOVERY - DATA SET

This appendix presents the information gathered from the exploratory study prior discussed in Chapter 5. The set of instruments included in this Appendix comprise the following forms: Appendix C.1 presents the **consent form** subjects must be given and signed before joining the Experimental Study, confirming permission to participate in the research; then Appendix C.2 details the **background questionnaire**, intended to collect data about the subjects background;

C.1 CONSENT FORM

Table C.1 presents the consent form used in the exploratory study.

C.2 BACKGROUND QUESTIONNAIRE

We designed a form to gather background information regarding their programming experience. Although the target SPL projects were written in Java, we also included questions about programming experience in other languages. Table C.2 shows a sample of the questionnaire the students filled out.

Table C.3 presents the exploratory study subjects profile. The age of the subjects varies from 23 to 47 years. There are subjects with experience in industry development (from 2 to 30 years). Regarding the experience with SPL, the subjects started to study the subject. Finally, the same pattern happens with experience with software architecture. Due to the relevance of the topic, we expected a stronger relation with software architecture. However, even the most experienced subjects did not use SA in your work daily basis.

Table C.4 shows the subjects experience with the programming languages. Java is the most popular programming language among the subjects because it is commonly used in industry. Moreover, the majority of subjects also worked with PHP. Finally, C and C++ are constantly used.

CONSENT FORM

Subject Name:

The information contained in this form is intended to establish a written agreement, whereby the student authorizes his/her participation in the exploratory study, with full knowledge of the nature of the procedures he/she will submit as a participant, with free will and without any duress. This participation is voluntary and the subject is free to withdraw from the study at any time and no longer participate in the study without prejudice to any service that is being or will be submitted.

I. STUDY TITLE:

On the behavior of the SAVaR.

II. STUDY GOAL:

Evaluate the accuracy and reliability of SAVaR.

III. RESPONSIBLE INSTITUTION:

Federal University of Bahia (UFBA).

IV. RESPONSIBLE RESEARCHERS:

Crescencio Lima, Ph.D. Candidate (UFBA), Ivan do Carmo Machado, Ph.D. (UFBA), and Christina von Flach Garcia Chavez, Ph.D. (UFBA).

V. CONSENT:

By signing this consent form, I certify that have read the information above and I am sufficiently informed of all statements, I fully agree to participate on the experiment. So, I authorize the execution of the research discussed above.

Salvador, BA, Brazil, / /

Signature

Table C.1 Consent Form

Table C.2 Characterization form used

ID	Java
Age	Integer
Have you worked as a software developer in academy?	Y/N
If Yes, How long did you worked?	Integer
Have you worked as a software developer in industry?	Y/N
If Yes, How long did you worked?	Integer
What are the programming languages your worked?	Text
Did you have previous experience developing SPL projects?	Y/N
If Yes, How long did you worked?	Integer
Did you have previous experience working with software architecture?	Y/N
If Yes, How long did you worked?	Integer

Figure C.1 shows the SPL Web Store project Design Structure Matrix (DSM). We provide the DSM details in the project website¹. Moreover, in the project website can be found the recovered PLA (module view), Metrics report, and the inspection report.

Figure C.2 presents the recovered PLA module view of the Web Store project.

Figure C.3 shows the SPL Message project Design Structure Matrix (DSM). We provide the DSM details in the project website². Moreover, in the project website can be found the recovered PLA (module view), Metrics report, and the inspection report.

Figure C.4 presents the recovered PLA module view of the Web Store project.

¹<<http://homes.dcc.ufba.br/~crescencio/WebStoreSPL/>>

²<<http://homes.dcc.ufba.br/~crescencio/SPLMessage/>>

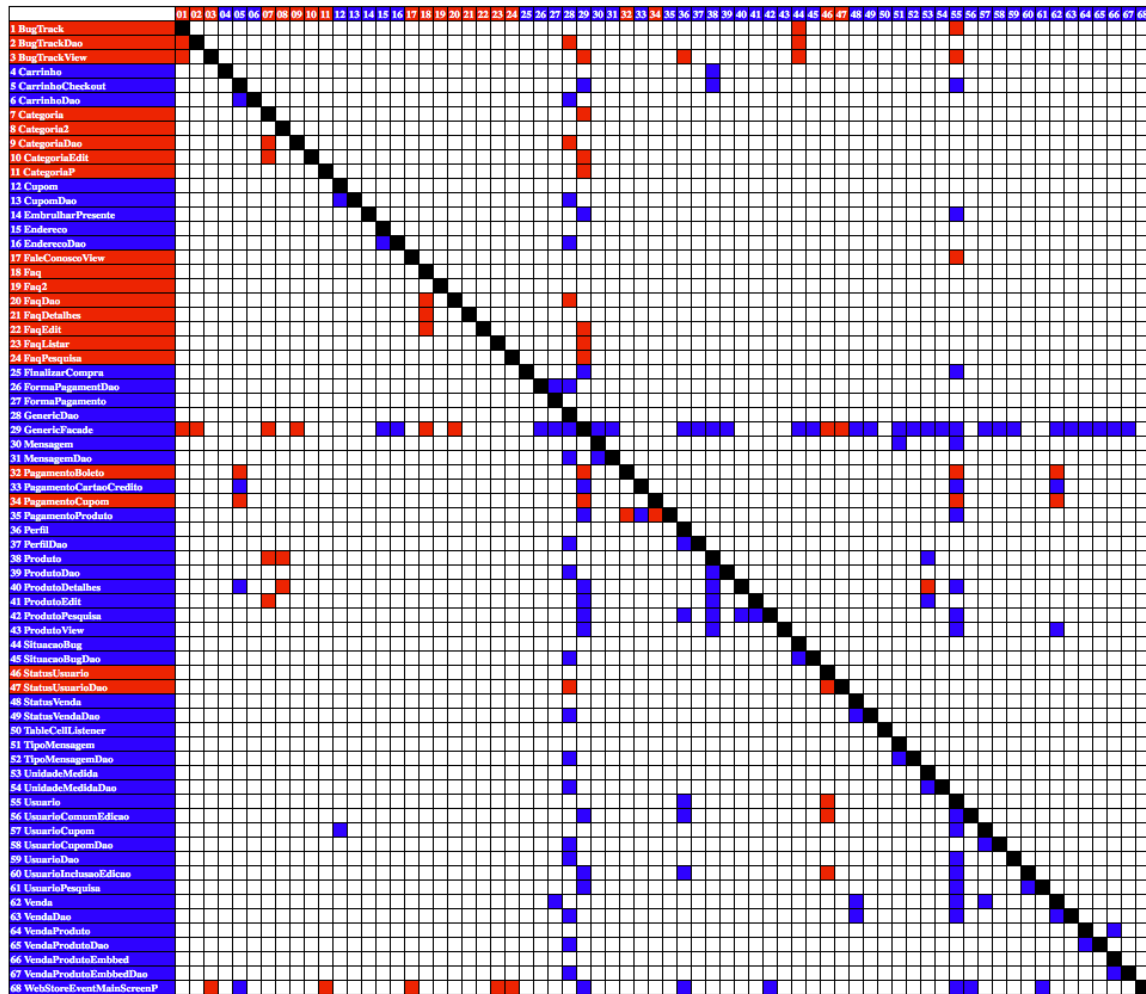


Figure C.1 Recovered Design Structure Matrix - Project SPL Web Store

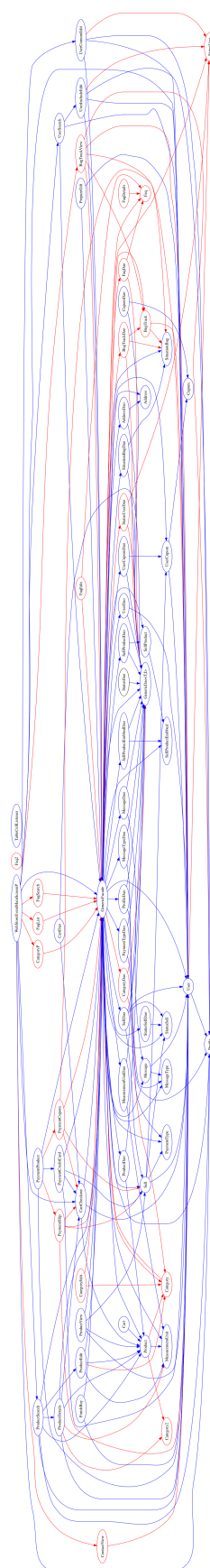


Figure C.2 Recovered PLA (module view) - Project SPL Web Store

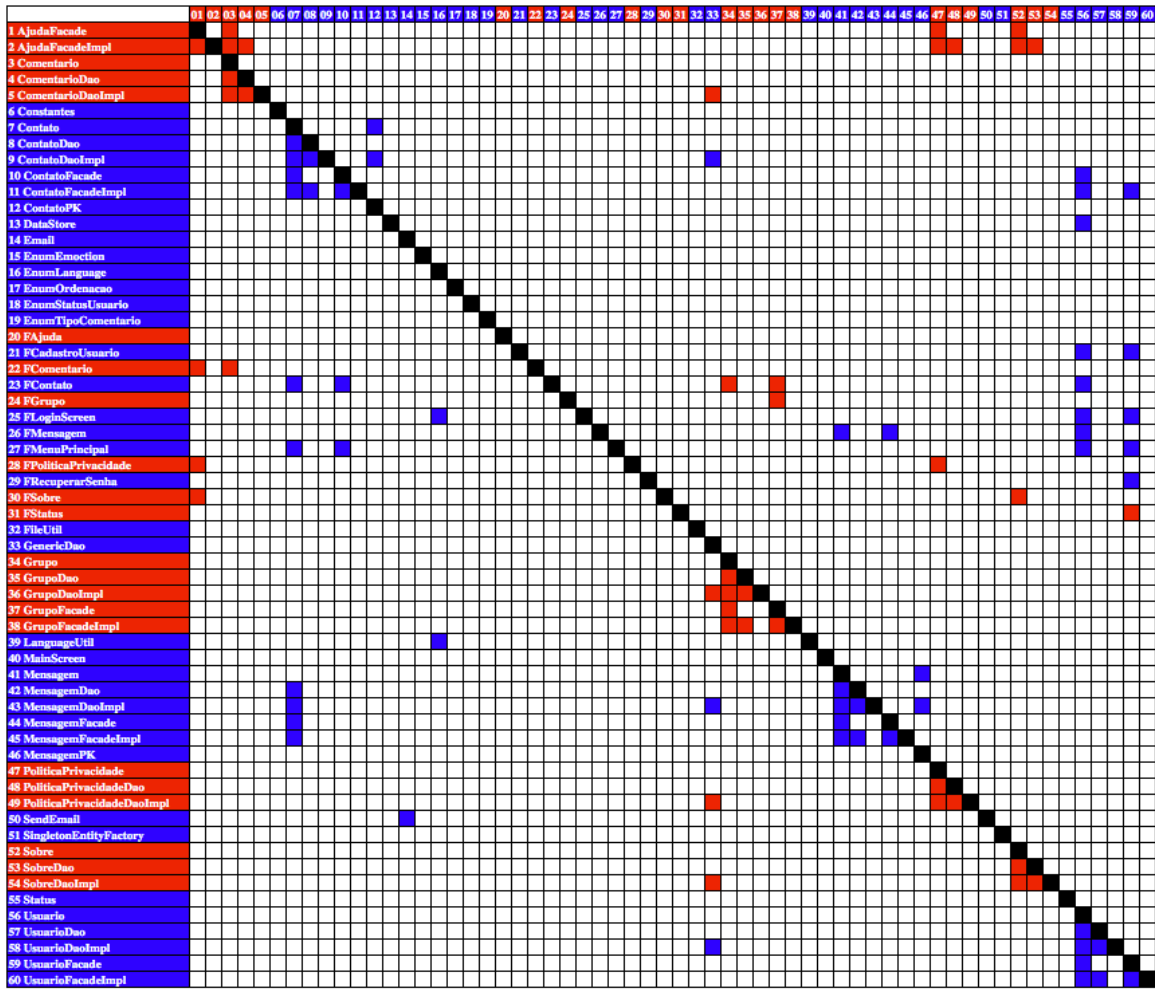


Figure C.3 Recovered Design Structure Matrix - Project SPL Message

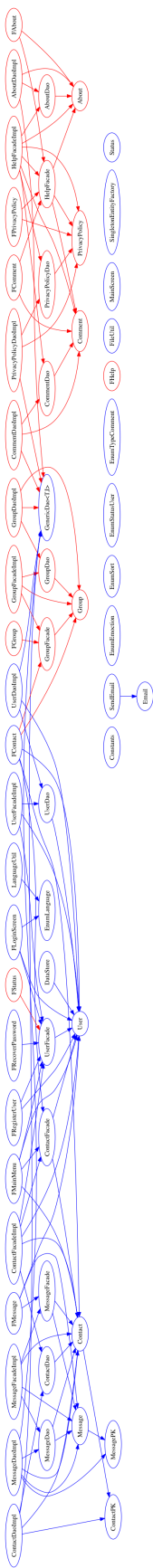


Figure C.4 Recovered PLA (module view) - Project SPL Message

Table C.3 Subjects' Profile

ID	Age	Gender	Exp. Ind.	Exp. Acad.	Exp. SPL	Exp. Arch.
1	23	masc.	2 yrs	5 yrs	3 mths	study
2	47	masc.	30 yrs	-	4 mths	study
3	23	masc.	1 mth	4 mths	study	study
4	27	masc.	3 yrs	2 yrs	study	2 yrs
5	24	fem.	2 yrs	3 yrs	6 mths	study
6	40	fem.	6 yrs	8 yrs	none	study
7	33	fem.	15 yrs	5 yrs	1 yr	study
8	31	masc.	3 yrs	-	6 mths	study
9	25	masc.	3 yrs	3 yrs	2 mths	2 yrs

Legend:[Exp. Ind.] Experience in industry, [Exp. Acad.] Experience in academia, [Exp. SPL] Experience in SPL, [Exp. Arch.] Experience in software architecture, [masc.] masculine, [fem.] feminine, [yrs] years, [mths] months

Table C.4 Subjects Experience with Programming Language

ID	Java	PHP	Pascal	Ruby	C	C++	C#	ASP	Other
1	•	•	-	•	-	-	-	-	•
2	-	•	•	-	-	-	-	•	•
3	-	•	-	-	-	•	-	-	-
4	•	-	-	-	•	•	•	-	•
5	•	•	-	-	•	-	-	-	-
6	•	-	•	-	•	-	-	-	•
7	•	•	-	-	-	-	•	•	•
8	•	•	•	-	•	•	-	-	-
9	•	-	-	-	•	•	-	-	-

Legend:[•] subject with professional experience using the programming language, [-] no experience with the programming language