



**Programa Multiinstitucional de Pós-Graduação em Ciência
da Computação- PMCC**

**UNDERSTANDING AND GUIDING SOFTWARE
PRODUCT LINES EVOLUTION BASED ON
REQUIREMENTS ENGINEERING ACTIVITIES**

By

Raphael Pereira de Oliveira

Ph.D. Thesis

SALVADOR
September/2015

PMCC-Dsc-0022

RAPHAEL PEREIRA DE OLIVEIRA

**UNDERSTANDING AND GUIDING SOFTWARE
PRODUCT LINES EVOLUTION BASED ON
REQUIREMENTS ENGINEERING ACTIVITIES**

Ph.D. Thesis presented to the Multi-institutional Graduate Program in Computer Science at Federal University of Bahia, Salvador University, and Feira de Santana State University in partial fulfillment of the requirements for the degree of Philosophy Doctor in Computer Science.

Advisor: Eduardo Santana de Almeida

SALVADOR
September/2015

Oliveira, Raphael Pereira de
O48u Understanding and Guiding Software Product Lines Evolution based on Requirements Engineering Activities / Raphael Pereira de Oliveira. – Salvador: Universidade Federal da Bahia, 2015.

237p. il.

Inclui apêndices e bibliografia.
Orientador: Prof. Dr. Eduardo Santana de Almeida.
Tese (doutorado) – Universidade Federal da Bahia, Instituto de Matemática, Universidade Salvador, Universidade Estadual de Feira de Santana, 2015.

1. Software Engineering. 2. Software Product Lines. 3. Software Evolution. 4. Requirements Engineering. 5. Empirical Studies. I. Almeida, Eduardo Santana de. II. Universidade Federal da Bahia, Instituto de Matemática. III. Universidade Salvador. IV. Universidade Estadual de Feira de Santana. V. Título.

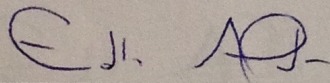
CDU – 004.41

RAPHAEL PEREIRA DE OLIVEIRA

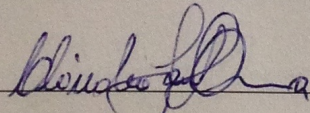
**" UNDERSTANDING AND GUIDING SOFTWARE PRODUCT LINES
EVOLUTION BASED ON REQUIREMENTS ENGINEERING ACTIVITIES "**

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

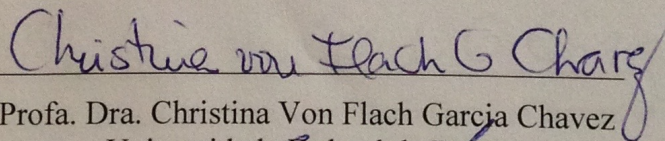
Salvador, 10 de setembro de 2015.



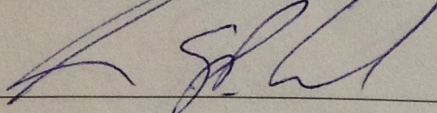
Prof. Dr. Eduardo Santana de Almeida
Universidade Federal da Bahia



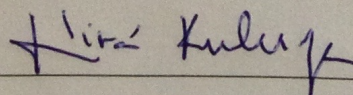
Prof. Dr. Cláudio Nogueira Sant'Anna
Universidade Federal da Bahia



Profª. Dra. Christina Von Flach Garcia Chavez
Universidade Federal da Bahia



Prof. Dr. Leonardo Gresta Paulino Murta
Universidade Federal Fluminense



Prof. Dr. Uirá Kulesza
Universidade Federal do Rio Grande do Norte

*To my beloved parents, Alzido de Oliveira (in memoriam)
and Francisca I. P. de Oliveira.*

Acknowledgements

First of all, I would like to thank God, who always blessed my way over this Ph.D.

Secondly, I would like to thank (from the bottom of my heart) to my family, mainly my beloved parents Alzido de Oliveira (*in memoriam*) and Francisca Isaldite Pereira de Oliveira. Dear Parents, all of this would not be possible without the encouragement and support that you have given to me. Father, I am very proud of the path that you took on this earth, spreading your knowledge all around the world. You are my inspiration and I know that, right now, you are happy with this moment, wherever you are. Mother, I see you as the greatest woman in the world who always knew how to overcome obstacles. Your heart, filled with kindness and dedication, taught me the true meaning of life, the love. I cannot forget the support that I had from my dear siblings over this Ph.D., thanks Fabíola, Sérgio, Denise, Ângela and Sônia!

My thanks to a very special person in my life, my fiancée Isabelle Aparecida Dellela Blengini. I am very happy to have you by my side whenever and wherever we are (São Carlos - SP, Salvador - BA, Valencia - Spain, and in a near future, Sergipe). Thanks for all your patience and support within the hard moments of my Ph.D., you gave me courage to deal with all of them. Thank you for showing me that together we are stronger to face the life's obstacles.

My sincere thanks to my advisor Eduardo Santana de Almeida. Eduardo, you showed me that we can always do more and also that the challenges are here to be faced. I admire your way of life as advisor and researcher. Many thanks for the growth opportunities that you offered to me. Thanks for the talks, discussions, and challenges that we faced together. Today, thanks to you, I consider myself much more mature and prepared for the world.

I also would like to thank the Professors Emilio Insfrán and Silvia Abrahão, from Polytechnic University of Valencia, in Spain. Thanks for hosting me in your research group during my Ph.D. sandwich. Our meetings, discussions, and presentations resulted (and continue to give results) on publications. Thank you for the opportunity to work with you.

Thanks to the collaboration, advice, and experiences that I could share with some Professors. Gecynalda Soares Silva Gomes, thanks to your statistics analysis, our work earned much more importance. Professor David Weiss, I always thanks for having had the opportunity to meet such a good person in my life, a great professional and a great person. And I cannot forget to thank the dear Joanne Weiss (David Weiss wife). Thank you very much for sharing with me your huge experience of life. Professor Nenad Medvidovic, my thanks for the advice and experience of life that we shared when you visited us in Bahia.

I cannot forget to thank the staff from CEAPG-MAT/UFBA. Thanks for all of your attention and availability, reserving rooms and attending to my requests, over these four years and a half.

Thank you very much Davilene, Solange, Márcio, Gustavo and Kleber!

I really thank to my dear friends of this journey that are, or have been, part of RiSE Labs, INES Laboratory (UFBA), Software Engineering Laboratory (LES-UFBA) and Department of Computer Systems and Computation (DSIC-Spain). It was very grateful to count on you in several discussions, meetings, and why not to mention, our entertainments moments. Many thanks to Ivan Machado, Alcemir Santos, Iuri Souza, Tassio Vale, Ivonei Freitas, Crescêncio Lima, Simone Morais, Thiago Souto, Paulo Silveira, Pdraig O’Leary, Carlos Andrade, Javier Gonzalez–Huerta, David Blanes, Priscila Cedillo, Kamil Krynicki, Luanna Lobato, Yguaratã Cavalcanti, Renato Novais, Rodrigo Rocha, Bruno Carreiro, Jonatas Bastos, Leandro Oliveira, Larissa Rocha, Michelle Larissa, Lorenzo Alvim, Magno Luã, Hugo Sica, Douglas Barbosa, Alex Bruno, Bruno Cabral, Karla Malta, Anna Luiza, Matheus Lessa e Williams Barbosa.

Life brings you some lovely friends, and I had the opportunity to meet two very special friends: Aidil Almeida and Luciano Melo. “Aunt” Aidil, you are a very special person that appeared in my life, you show the city of Salvador to me, all its culture and foods. You are a very special friend of my that will live in my heart forever! Luciano Melo, it is amazing to meet a person with a huge heart like yours. I will never forget all the support that you gave me when I arrived at Estância, Sergipe. Thanks so much my dear friend. Our friendship will live forever!

To the committee of my Ph.D. defense, my sincere thanks. All of your questions and suggestions helped to improve this Thesis. Thanks a lot Leonardo Murta, Uirá Kulesza, Christina Von Flach, Cláudio Sant’Anna and Eduardo Almeida.

Again, a very special thanks to my family that was at my defense giving me an special support. Thanks a lot Francisca Isaldite, Isabelle Blengini, Fabíola Oliveira, Letícia Macedo, Pedro Henrique, Antônio Leite, Helena Repolho, and Yara Matos.

Thanks for the financial support of FAPESB - Foundation of the Bahia State Research, CAPES, and CAPES-DGU. Without these scholarships, this investigation could not be performed. I also thank you the Federal Institute of Sergipe (IFS), mainly Gino, Fernando, and Lunalva, for the financial support on traveling expenses to present a paper abroad. My thanks to Patrícia da Silva Santos for helping me with the catalog in publication.

Finally, thanks to all the others who have crossed my way over these four and a half years of my Ph.D. “*And in the end, the*” product “*you take, is equal to the*” work, “*you make!*” Thus, let’s keep working!

Agradecimentos

Em primeiro lugar a Deus, que sempre abençoou meu caminho durante esse doutorado.

Em segundo lugar, agradeço do fundo do meu coração a minha família, especialmente meus queridos pais Alzido de Oliveira (*in memoriam*) e Francisca Isaldite Pereira de Oliveira. Pais, saibam que tudo isso não seria possível sem o incentivo e apoio que vocês me deram. Pai, tenho muito orgulho do caminho que você trilhou aqui na terra e dos diversos frutos que você plantou, você é minha inspiração de vida e sei que estás feliz com esse momento, queira Deus onde você estiver. Mãe, saiba que vejo em você uma mulher guerreira que sempre soube superar qualquer obstáculo. Seu coração repleto de bondade e dedicação me ensinou o verdadeiro sentido da vida, o amor. Não posso deixar de fora o apoio que tive dos meus irmãos em diversos momentos desse doutorado, muito obrigado Fabíola, Sérgio, Denise, Ângela e Sônia!

O meu muito obrigado também a uma pessoa muito especial em minha vida, a minha noiva Isabelle Aparecida Dellela Blengini. Fico muito feliz por ter você do meu lado sempre, seja onde for, São Carlos – SP, Salvador – BA, Valência – Espanha, e futuramente em Sergipe. Obrigado pela paciência nos momentos difíceis do meu doutorado e por todo suporte, principalmente o emocional, que você me deu. Obrigado por me mostrar que juntos somos mais fortes para enfrentar qualquer obstáculo.

Meus sinceros agradecimentos ao meu orientador Eduardo Santana de Almeida. Eduardo, você me mostrou que sempre podemos mais e que os desafios estão aqui para serem enfrentados. Admiro muito seu trajeto de vida como orientador e pesquisador. Meu muito obrigado pelas diversas oportunidades de crescimento que você me ofereceu. Graças as diversas conversas, discussões e desafios que enfrentamos juntos, hoje me considero muito mais maduro e preparado para o mundo.

Gostaria de agradecer também aos professores Emilio Insfrán e Silvia Abrahão da Universidade Politécnica de Valência, na Espanha. Obrigado pelo carinhoso acolhimento na Espanha durante meu doutorado sanduíche. Nossas reuniões, discussões e apresentações renderam (e continuam rendendo) diversos frutos. Obrigado pela oportunidade de trabalhar com vocês.

Obrigado também aos trabalhos, conselhos e experiências que pude partilhar com alguns professores. Gecynalda Soares Silva Gomes, graças a sua estatística, nossos trabalhos ganharam muito mais relevância. Professor David Weiss, agradeço sempre por ter tido a oportunidade de encontrar uma pessoa tão boa como você em minha vida, um excelente profissional e uma excelente pessoa, sempre disposto a ajudar. E não poderia deixar de agradecer a querida Joanne Weiss (esposa do professor David Weiss). Muito obrigado por compartilhar a enorme experiência de vida de vocês. Ao professor Nenad Medvidovic, o meu muito obrigado pelos conselhos e

experiência de vida que partilhamos no período que esteve conosco na Bahia.

Não posso me esquecer dos funcionários do CEAPG-MAT/UFBA. Meu muitíssimo obrigado pela atenção e disponibilidade, nas reservas de salas e solicitações, ao longo desses pouco mais de quatro anos. Muito obrigado Davilene, Solange, Márcio, Gustavo e Kleber!

Agradeço aos meus amigos e parceiros dessa caminhada que fazem ou fizeram parte do RiSE Labs, Laboratório INES (UFBA), Laboratório de Engenharia de Software (LES-UFBA) e Departamento de Sistemas Informáticos Y Computación (DSIC-Espanha). Muito grato de poder contar com vocês em diversas discussões, reuniões e por que não, diversões. O meu muito obrigado a Ivan Machado, Alcemir Santos, Iuri Souza, Tassio Vale, Ivonei Freitas, Crescêncio Lima, Simone Morais, Thiago Souto, Paulo Silveira, Pdraig O’Leary, Carlos Andrade, Javier Gonzalez–Huerta, David Blanes, Priscila Cedillo, Kamil Krynicki, Luanna Lobato, Yguaratã Cavalcanti, Renato Novais, Rodrigo Rocha, Bruno Carreiro, Jonatas Bastos, Leandro Oliveira, Larissa Rocha, Michelle Larissa, Loreno Alvim, Magno Luã, Hugo Sica, Douglas Barbosa, Alex Bruno, Bruno Cabral, Karla Malta, Anna Luiza, Matheus Lessa e Williams Barbosa.

A vida nos traz amigos muito amáveis, e eu tive a oportunidade de encontrar duas delas: Aidil Almeida e Luciano Melo. “Tia” Aidil, você é uma pessoa especial que apareceu na minha vida. Você me apresentou Salvador, sua cultura e culinária. Você é uma amiga muito especial que irá morar no meu coração para sempre! Luciano Melo, é muito legal encontrar uma pessoa com um coração tão grande como o seu. Eu nunca vou me esquecer do suporte que você me deu quando eu cheguei em Estância, Sergipe. Muito obrigado meu querido amigo. Nossa amizade será eterna!

À banca do meu doutorado, meus sinceros agradecimentos. Todas as questões e sugestões levantadas ajudaram a melhorar essa Tese. Muito obrigado Leonardo Murta, Uirá Kulesza, Christina Von Flach, Cláudio Sant’Anna e Eduardo Almeida.

Mais uma vez, um muito obrigado a minha família que estava presente na minha defesa me dando um suporte adicional! Muito obrigado Francisca Isaldite, Isabelle Blengini, Fabíola Oliveira, Letícia Macedo, Pedro Henrique, Antônio Leite, Helena Repolho e Yara Matos.

Agradeço o apoio financeiro da FAPESB - Fundação de Amparo a Pesquisa do Estado da Bahia, CAPES, CAPES-DGU, sem as bolsas, em seus respectivos momentos, não seria possível realizar este trabalho. Eu também agradeço ao Instituto Federal de Sergipe (IFS), principalmente Gino, Fernando e Lunalva, pelo apoio financeiro relativo a apresentação de um artigo no exterior. Muito obrigado Patrícia da Silva Santos por me ajudar com a ficha catalográfica da minha Tese.

Para finalizar, o meu muito obrigado a todos os demais que passaram pelo meu caminho nesses quatro anos e meio do doutorado. No final, a colheita é proporcional ao que foi plantado. Por isso, continuemos plantando!

*“And in the end
The love you take
Is equal to
The love you make”*

—JOHN LENNON AND PAUL MCCARTNEY (THE BEATLES)

Abstract

Software Product Line (SPL) has emerged as an important strategy to cope with the increasing demand of large-scale products customization. SPL has provided companies with an efficient and effective means of delivering products with higher quality at a lower cost, when compared to traditional software engineering strategies. However, such benefits do not come for free.

There is a necessity in SPL to deal with the evolution of its assets to support changes within the environment and user needs. These changes in SPL are firstly represented by requirements. Thus, SPL should manage the commonality and variability of products by means of a “*Requirements Engineering (RE) - change management*” process. Hence, besides dealing with the reuse and evolution of requirements in an SPL, the RE for SPL also needs an approach to represent explicitly the commonality and variability information (*e.g.*, through feature models and use cases).

To understand the evolution in SPL, this Thesis presents two empirical studies within industrial SPL projects and a systematic mapping study on SPL evolution. The two empirical studies evaluated Lehman’s laws of software evolution in two industrial SPL projects, demonstrating that most of the laws are supported by SPL environments. The systematic mapping study on SPL evolution identified approaches in the area and revealed gaps for researching, such as, that most of the proposed approaches perform the evolution of SPL requirements in an *ad-hoc* way and were evaluated through feasibility studies.

These results led to systematize, through guidelines, the SPL processes by starting with the SPL requirements. Thus, it was proposed an approach to specify SPL requirements called *Feature-Driven Requirements Engineering (FeDRE)*. FeDRE specifies SPL requirements in a systematic way driven by a feature model. To deal with the evolution of FeDRE requirements, a new approach called *Feature-Driven Requirements Engineering Evolution (FeDRE²)* was presented. FeDRE² is responsible for guiding, in a systematic way, the SPL evolution based on activities from RE. These two proposed approaches are responsible for dealing with the first phase of the SPL development, which is the RE for the domain engineering.

Both proposed approaches, FeDRE and FeDRE², were evaluated and the results, besides being preliminaries, shown that the approaches were perceived as easy to use and also useful, coping with the improvement and systematization of SPL processes.

Keywords: Software Product Lines, Software Evolution, Requirements Engineering, Empirical Studies.

Table of Contents

List of Figures	xxi
List of Tables	xxiii
List of Acronyms	xxv
I Introduction	1
1 Introduction	3
1.1 Motivation	4
1.2 Objective	5
1.3 Research Method	6
1.4 Research History	8
1.5 Contributions	11
1.6 Out of Scope	12
1.7 Organization of the Thesis	13
II Background	17
2 Background	19
2.1 Software Product Lines	19
2.1.1 Software Product Line Essential Activities	20
2.1.2 Software Product Line Variability Management	22
2.2 Software Product Lines Evolution	25
2.2.1 Forces for Change	25
2.2.2 Evolution Propagation	26
2.3 Software Product Lines Requirements Engineering	27
2.3.1 Risks and Challenges	30
2.4 Evolution of Software Product Lines Requirements	31
2.5 Chapter Summary	32

III	Understanding Software Product Lines Evolution	33
3	Empirical Studies on the Application of Lehman’s Laws within the Industry	35
3.1	Introduction	35
3.2	Related Work	37
3.3	Empirical Studies	40
3.3.1	General Planning	40
3.3.2	First Empirical Study	42
3.3.2.1	Execution	43
3.3.2.2	Data Analysis and Discussion	45
3.3.3	Motivation for Conducting the Replication	51
3.3.4	Changes to the Original Experiment	51
3.3.5	Second Empirical Study	52
3.3.5.1	Execution	53
3.3.5.2	Data Analysis and Discussion	54
3.3.6	Comparison and Discussion of Results	59
3.3.6.1	Consistent Results	60
3.3.6.2	Partially Consistent Results	60
3.3.6.3	Partially Different Results	60
3.3.6.4	Differences in Results	61
3.4	Threats to Validity	62
3.5	Key Findings and Contributions for SPL Community	63
3.6	Chapter Summary	65
4	Software Product Lines Evolution: A Systematic Mapping Study	67
4.1	Introduction	67
4.1.1	Motivation	68
4.2	Background	68
4.2.1	Related Work	69
4.3	Research Method	70
4.3.1	Planning Stage	70
4.3.1.1	Research Question	70
4.3.1.2	Search Strategy	70
4.3.1.3	Selection of primary studies	73
4.3.1.4	Quality Assessment	74

4.3.1.5	Data Extraction Strategy	75
4.3.2	Conducting Stage	78
4.3.2.1	Search from 1996 up to 2014	78
4.3.2.2	Selection of Studies from 1996 up to 2014	80
4.4	Results	80
4.4.1	Why SPL approaches need to deal with evolution? (RQ1.1)	84
4.4.2	When the SPL approaches perform the evolution? (RQ1.2)	86
4.4.3	Where the SPL approaches perform the evolution? (RQ1.3)	89
4.4.4	What type of Evolution (static or dynamic) does the approach support? (RQ1.4)	93
4.4.5	How SPL approaches support the evolution? (RQ1.5)	95
4.4.6	What is the SPL life cycle and phase in which the evolution is applied? (RQ1.6)	97
4.4.7	What is the evaluation procedure from the approach? (RQ1.7)	99
4.4.8	What type of tool support does the approach offer? (RQ1.8)	100
4.4.9	In which context the approach is applied? (RQ1.9)	100
4.4.10	Mapping Results	100
4.4.11	Threats to Validity	111
4.5	Chapter Summary	112

IV Guiding Software Product Lines Evolution based on Requirements Engineering Activities 115

5	Feature-Driven Requirements Engineering (FeDRE) Approach	117
5.1	Introduction	117
5.2	Related Work	119
5.3	Feature-Driven Requirements Engineering Approach For SPL	121
5.3.1	Scoping	122
5.3.1.1	Existing Assets	123
5.3.1.2	Feature Model	123
5.3.1.3	Feature Specification	124
5.3.1.4	Product Map	124
5.3.2	Requirements Specification for Domain Engineering	125
5.3.2.1	Glossary	126

5.3.2.2	Functional Requirements	126
5.3.2.3	Traceability Matrix	127
5.3.3	Guidelines for Specifying SPL Functional Requirements	127
5.4	Empirical Study	130
5.4.1	Design of the empirical study	130
5.4.2	Preparation of the empirical study	134
5.4.3	Collection of the data	136
5.4.3.1	Which features can be grouped to be specified by UC?	136
5.4.3.2	What are the specific UC for the feature or set of features?	137
5.4.3.3	Where the UC should be specified?	137
5.4.3.4	How each UC is specified in terms of steps?	138
5.4.4	Data Analysis	138
5.4.4.1	First Quantitative Analysis	139
5.4.4.2	Second Quantitative Analysis	140
5.4.5	Threats to validity	141
5.5	Chapter Summary	142
6	Feature-Driven Requirements Engineering Evolution (FeDRE²) Approach	143
6.1	Introduction	143
6.2	Background	144
6.2.1	Related Work	145
6.3	FeDRE ² Approach	147
6.3.1	Task 1: Identify the Evolution Scenario	148
6.3.1.1	Change Request Artifact	148
6.3.1.2	Feature Model Artifact	149
6.3.1.3	Use Case Textual Specification Artifact	150
6.3.1.4	Use Case Diagram Artifact	150
6.3.1.5	Product Map Artifact	150
6.3.2	Task 2: Evolve the SPL Requirements	152
6.3.3	Task 3: Update the Traceability Matrix	154
6.3.3.1	Traceability Matrix Artifact	154
6.4	Empirical Study	156
6.4.1	Design of the Empirical Study	156
6.4.2	Preparation of the Empirical Study	158
6.4.3	Data Collection	160

6.4.3.1	Background Form	160
6.4.3.2	Empirical Study	160
6.4.3.3	Survey	161
6.4.4	Data Analysis	162
6.4.4.1	First Quantitative Analysis	162
6.4.4.2	Second Quantitative Analysis	163
6.4.5	Threats to Validity	167
6.5	Chapter Summary	168
V	Conclusions and Future Work	169
7	Conclusions	171
7.1	Future Work	172
7.1.1	<i>Evaluating Lehman’s Laws (LL) of Software Evolution</i>	172
7.1.2	<i>Systematic Mapping Study on SPL Evolution</i>	172
7.1.3	<i>Feature-Driven Requirements Engineering (FeDRE) Approach</i>	173
7.1.4	<i>Feature-Driven Requirements Engineering Evolution (FeDRE²) Approach</i>	173
7.2	Related Work	173
7.3	Main Contributions	174
	References	176
	Appendices	187
A	Empirical Studies	189
A.1	The KPSS Test and Hypotheses results (at MC)	190
A.2	The KPSS Test and Hypotheses results (at FC)	191
B	SPL Evolution: A Systematic Mapping Study	193
B.1	Primary studies selected	194
B.2	Data Extraction Form	208
B.3	Search String for each Electronic Database	212
B.4	Mapping of the primary studies	214
C	Feature-Driven Requirements Engineering (FeDRE) Approach	223
C.1	Survey Statements to Evaluate FeDRE, based on PEOU and PU variables	224

C.2	Identified Use Cases for each Feature	225
C.3	Subject's Responses for PEOU and PU	226
C.4	Box Plots for PEOU and PU Variables	227
D	Feature-Driven Requirements Engineering Evolution (FeDRE²) Approach	229
D.1	Survey Statements to Evaluate FeDRE ² , based on PEOU and PU variables . . .	230
D.2	FeDRE ² Background Form	231
D.3	FeDRE ² Survey	234
D.4	Subject's Responses for PEOU and PU	237

List of Figures

1.1	Thesis Motivation	5
1.2	Research Methodology	7
1.3	Research Timeline	9
1.4	Ph.D. Thesis Organization	14
2.1	Software Product Lines Essential Activities (Clements and Northrop, 2002)	20
2.2	SPL Core Asset Development (Clements and Northrop, 2002)	21
2.3	SPL Production Plan (Clements and Northrop, 2002)	22
2.4	SPL Product Development (Clements and Northrop, 2002)	23
2.5	Feature Model Example	24
2.6	SPL Assets Types	24
3.1	Modules (assets) per Areas Supported by MC.	44
3.2	Plotted Graphs from CIC data and LOC (at MC)	46
3.3	Confidence Intervals for the Regression Coefficients (at MC)	48
3.4	Modules (assets) per Area at FC.	53
3.5	Plotted Graphs from Bug Tracking System data and LOC (at FC)	55
3.6	Confidence Intervals for the Regression Coefficients (at FC)	56
4.1	Paper Selection Process	81
4.2	Bubble Chart for the combination of RQ1.1 (Why) by RQ1.6 (SPL Life Cycle) and RQ1.7 (Evaluation)	101
4.3	Bubble Chart for the combination of RQ1.1 (Why) by RQ1.8 (Tool) and RQ1.9 (Context)	102
4.4	Bubble Chart for the combination of RQ1.2 (When) by RQ1.6 (SPL Life Cycle) and RQ1.7 (Evaluation)	103
4.5	Bubble Chart for the combination of RQ1.2 (When) by RQ1.8 (Tool) and RQ1.9 (Context)	104
4.6	Bubble Chart for the combination of RQ1.3 (Where) by RQ1.6 (SPL Life Cycle) and RQ1.7 (Evaluation)	105
4.7	Bubble Chart for the combination of RQ1.3 (Where) by RQ1.8 (Tool) and RQ1.9 (Context)	106
4.8	Bubble Chart for the combination of RQ1.4 (What) by RQ1.6 (SPL Life Cycle) and RQ1.7 (Evaluation)	107

4.9	Bubble Chart for the combination of RQ1.4 (What) by RQ1.8 (Tool) and RQ1.9 (Context)	108
4.10	Bubble Chart for the combination of RQ1.5 (How) by RQ1.6 (SPL Life Cycle) and RQ1.7 (Evaluation)	109
4.11	Bubble Chart for the combination of RQ1.5 (How) by RQ1.8 (Tool) and RQ1.9 (Context)	110
5.1	Overview of the FeDRE approach.	122
5.2	Detailed Scoping Activity.	123
5.3	Product Map Example.	125
5.4	Detailed Requirements Specification Activity.	127
5.5	Meta-Model for SPL Requirements.	129
5.6	Overview of Activities, Tasks and Artifacts from the Guidelines.	130
5.7	Guidelines For Specifying SPL Functional Requirements.	131
5.8	Selected Features from the Feature Model for the Case Study.	135
5.9	UC Diagram (Feature Access Control).	138
6.1	SPEM Profiles used by FeDRE ²	145
6.2	FeDRE ² Approach Overview.	147
6.3	Change Request Example.	149
6.4	Feature Model Example.	149
6.5	Use Case Diagram Example.	151
6.6	Product Map Example.	151
6.7	Guidelines For Evolving SPL Functional Requirements.	153
6.8	Excerpt of a Traceability Matrix.	155
6.9	Perceived Ease of Use and Perceived Usefulness Box Plot.	165
6.10	Perceived Ease of Use Box Plot, for each Session (Brazil and Spain).	165
6.11	Perceived Usefulness Box Plot, for each Session (Brazil and Spain).	166

List of Tables

2.1	SPL Use Case Example.	29
3.1	Lehman’s Laws of Software Evolution (Herraiz <i>et al.</i> , 2013).	36
3.2	Relationship among Laws, Dependent Variables and Measurement, based on Barry <i>et al.</i> (2007).	42
3.3	Maintenance Types Groups at MC.	45
3.4	Laws and Results from the First Empirical Study (at MC).	51
3.5	Maintenance Types Groups at FC.	53
3.6	Laws and Results from the Second Empirical Study (at FC).	59
3.7	Consistent/Different Results from the Empirical Studies.	59
4.1	Sub-Research Questions and Motivation.	71
4.2	Selected Conferences for Manual Search.	72
4.3	Selected Journals for Manual Search.	72
4.4	Search String for the Digital Libraries.	73
4.5	Quality Assessment Form.	74
4.6	Total of Retrieved Papers from Databases (1996-2014)	79
4.7	Total of Retrieved Papers from the Manual Search on Conferences (1996-2014)	79
4.8	Total of Retrieved Papers from the Manual Search on Journals (1996-2014)	79
4.9	Total of Retrieved Papers (1996-2014)	79
4.10	Summary of the Results.	81
4.11	Association of the main findings with Why/When/Where/What/How SPLs Evolve.	112
4.12	Findings Consolidation	114
5.1	Comparative among current RE proposals from SPL.	120
5.2	Features Variability.	124
5.3	Planning.	133
5.4	Excerpt from the Glossary.	134
5.5	Excerpt from the Traceability Matrix.	137
5.6	Retrieve Contacts Use Case Specification.	139
5.7	Mean and Standard Deviation for the analyzed variables.	140
5.8	Analysis of the PEOU and PU variables.	141

6.1	Safe Evolution Templates, Evolution Scenarios and Evolution Scenarios Description.	148
6.2	Send Message Use Case Textual Specification Example.	150
6.3	Empirical Study Design.	158
6.4	Mean and Standard Deviation for the Analyzed Objective Dependent Variables.	163
6.5	Analysis of PEOU and PU Variables.	164
6.6	Mann-Whitney U Test Analysis of PEOU and PU Variables for Each Session.	167

List of Acronyms

AE	Application Engineering
CAD	Core Asset Development
DE	Domain Engineering
FeDRE	Feature-Driven Requirement Engineering
FeDRE²	Feature-Driven Requirement Engineering Evolution
FM	Feature Model
CR	Change Request
LL	Lehman's Laws
LOC	Lines Of Code
PD	Product Development
RE	Requirements Engineering
SPL	Software Product Lines

PART I

Introduction

1

Introduction

Advances in technologies, platforms and devices, which occur in various areas of computing, have required a great effort of the software engineering field in order to achieve goals such as: to decrease the software development costs, to cope with tight schedules, to deal with the market pressure, to maintain market presence, to improve the software development quality, to improve scale productivity, and to enable mass customization. In order to deal with these goals, software engineering offers paradigms and approaches to achieve a more effective software development.

One of the paradigms to achieve such goals is Software Product Lines (SPL), which is based on a set of systems sharing a common, managed suite of features that satisfy the specific needs of a particular market or mission. The products which compose the SPL are developed from a common set of core assets in a prescribed way (Clements and Northrop, 2002).

However, SPL demands approaches that deal with the evolution of its assets (common, variable and product specific assets) to support changes within the environment/user needs, and to keep its satisfactory performance (Mens and Demeyer, 2008). If the software does not support changes, then it will gradually lapse into uselessness (First Law of Software Evolution, Lehman, 1974). Sources of changes in SPL include changes: targeted to the entire product line; targeted to some products; and repositioning of assets from an individual product to the entire SPL (Svahnberg and Bosch, 1999; Ajila and Kaba, 2004; Bailetti *et al.*, 2004). Thus, the management of the SPL evolution is an essential activity to its success. Managing the changes in an SPL can also bring some benefits such as: to improve the traceability between artifacts in core assets and products (Ajila and Kaba, 2004); to avoid some irregular growth or decrease before it becomes a threat to the system; and to use the products feedback to improve the core asset.

These changes in SPL are firstly represented by requirements. Thus, an SPL has to manage the commonality and variability of products by means of a “*Requirements Engineering (RE)*” -

change management” process (Clements and Northrop, 2002). Besides dealing with the systematic reuse of requirements in an SPL, RE also needs to represent explicitly the commonality and variability information (*e.g.*, through feature models and use cases) (Alves *et al.*, 2010). Moreover, in SPL development, the requirements specification is even more critical (Clements and Northrop, 2002), since it is necessary to deal with common, variable, and product-specific requirements, not only for a single product but also for the whole set of products in the family.

Since SPL evolution is an essential activity and SPL RE is the basis of the SPL development, there is a need to understand and improve the software product lines evolution process, mainly for requirements. Thus, in this Thesis, studies were performed to understand the SPL evolution and approaches were proposed in order to deal with the specification and evolution of SPL requirements.

1.1 Motivation

The evolution of a software product line is risky because it can impact several products. Thus, when evolving an SPL it is important to make sure that the requirements of existing products are changed correctly. However, most of the approaches perform the SPL evolution only at the feature model level (Alves *et al.*, 2006; Thüm *et al.*, 2009; Botterweck *et al.*, 2010; Schulze *et al.*, 2012), as show in Figure 1.1 (ellipse A). Borba *et al.* (2012) also deal with SPL evolution at the feature model level considering configuration knowledge to trace features and SPL artifacts, however, how the SPL artifacts evolve, specially requirements, it is not taken into account, as show in Figure 1.1 (ellipses A and B). Neves *et al.* (2011) also deal with the evolution of SPL (features and assets, Figure 1.1 ellipses A and B), however, they also do not present how an asset should evolve. Thus, when SPL approaches perform the requirements engineering activity they lack of guidelines (Alves *et al.*, 2010) and conduct this activity in an *ad-hoc* way. The lack of guidance for performing the SPL requirements evolution activity makes this activity a tough task, since the requirement engineer will not have a catalog of possible evolution scenarios and actions to handle each evolution. Hence, guidelines for supporting the evolution of SPL features and requirements should exist to systematize and make more effective this process (Figure 1.1, ellipse C).

Thus, we decided to improve the SPL evolution by understanding how the SPL evolution process is performed, and by proposing two approaches for specifying and evolving SPL requirements in a systematic way, through guidelines (Figure 1.1, ellipses A, B, and C). The proposed approaches intend to make easier and more effective the SPL evolution process.

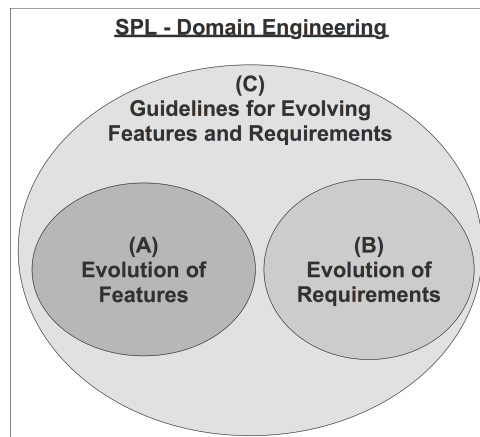


Figure 1.1: Areas Covered by this Thesis.

The choice of improving the SPL requirements evolution process was taken because requirements engineering is the basis of the SPL development and there is still a lack of guidelines for this process. It is important to perform an effective SPL RE activity and also handle its evolution. Managing the SPL RE evolution may bring some benefits, such as: to keep the SPL requirements rightly specified and updated; and to maintain traceability links between requirements and other artifacts.

Hence, considering the existing RE and evolution support for SPL engineering, and the SPL demands, the central problem addressed in this Thesis is the lack of adequate support for specifying and evolving SPL requirements in a systematic way through guidelines.

1.2 Objective

The main goal of this thesis is to improve SPL evolution based on requirements activities, through systematic guidelines.

The research has the following specific objectives:

- Understand, through empirical studies, the SPL evolution;
- Propose approaches for specifying and evolving SPL requirements according to guidelines;
- Perform empirical evaluations of the proposed approaches;

On the basis of such defined objectives, it was established the research question that drives this investigation:

How to achieve SPL requirements specification and evolution in a systematic way?

It is hypothesized that the proposed approaches, for specifying and evolving SPL requirements, are useful and facilitate the SPL RE specification and evolution. Moreover, the proposed approaches may increase the effectiveness within the tasks for specifying and evolving SPL requirements.

1.3 Research Method

This Section describes the research design employed as the basis for this Thesis. Based upon the research objectives, we decided to apply a combination of methods, to both gain a further understanding of the research problem, and to enrich our conclusions (Hesse-Biber, 2010).

The present research can be split into four specific parts: *1. Background*, *2. Understanding SPL Evolution*, *3. Guiding SPL Activities*, and *4. Evaluation*. Figure 1.2 shows a diagram with these macro parts and an overview of the sub-activities.

Part 1. Background

The background of this research was built based on relevant topics from the Software Engineering area. The studied topics included Software Product Lines (SPL), SPL Evolution, SPL RE, and Evolution of SPL requirements. Several papers and books aid in the construction of this background and also helped within the execution of the next part of the research.

Part 2. Understanding SPL Evolution

This second part is divided in two main group of activities, the industrial studies and the research studies to better understand the SPL evolution processes.

In the former, two exploratory empirical studies were performed within companies located in Salvador, Bahia, Brazil. These studies helped to understand better how the SPL assets evolve over a period of time.

In the latter, it was performed an SPL evolution mapping study. The SPL evolution mapping study helped to identify relevant approaches that deal with SPL evolution and possible gaps of research.

Part 3. Guiding SPL Activities

Based on the background and the knowledge acquired, it was identified a gap of approaches describing, in a systematic way, the specification and evolution of SPL requirements.

In order to deal with the specification of SPL requirements in a systematic way, it was

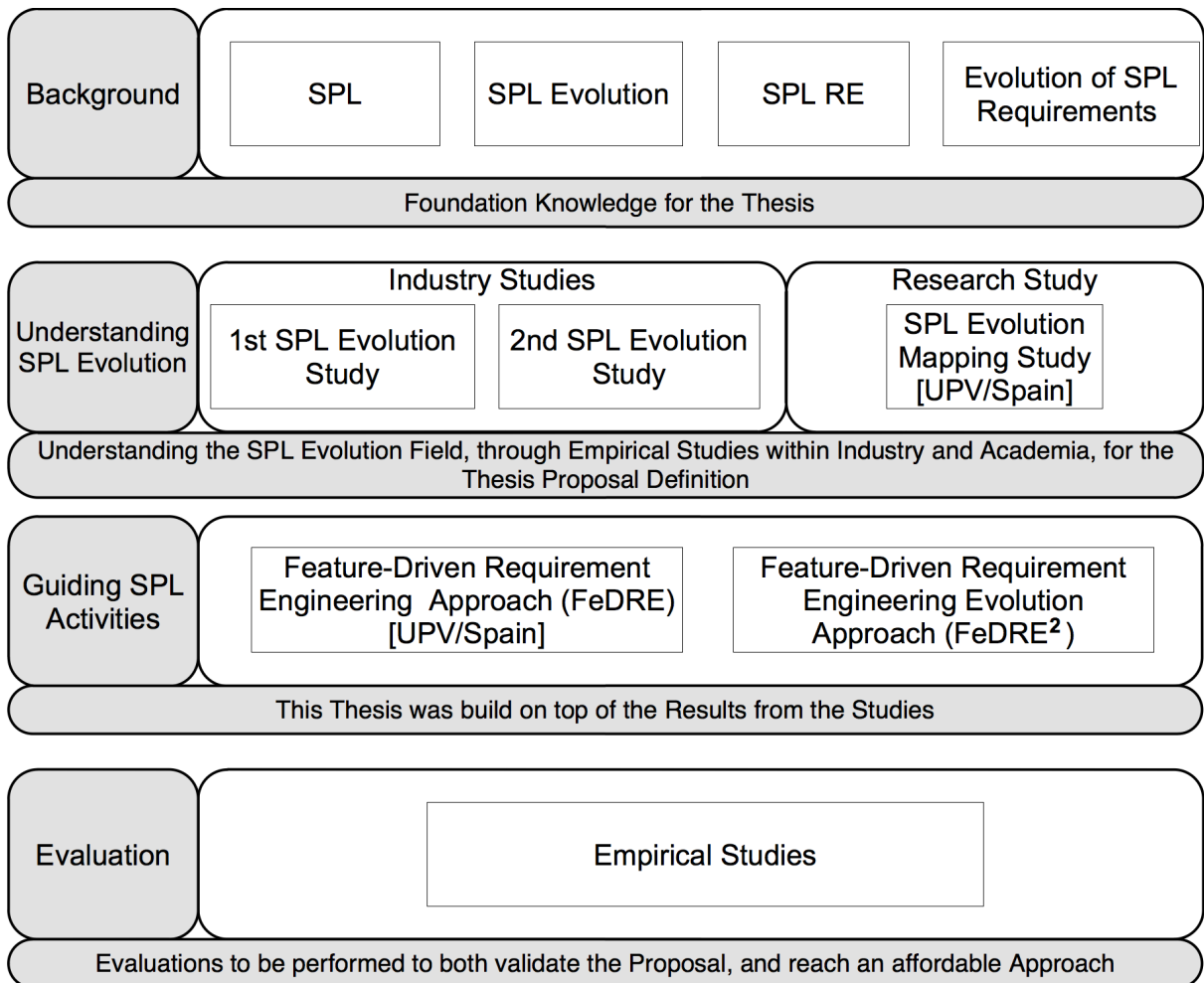


Figure 1.2: Research Methodology.

defined and evaluated a new SPL requirements specification approach called *Feature-Driven Requirements Engineering (FeDRE)*.

Thus, the following part of the research was to propose a *Feature-Driven Requirements Engineering Evolution (FeDRE²)* approach to deal with the SPL requirements evolution in a systematic way.

Part 4. Evaluation

The last part of this work consisted of carrying out the evaluation and refinement of the proposed approaches. In order to evaluate both approaches, an empirical study with Brazilian and Spanish subjects was performed for each approach. For each approach, we evaluated the perceived easy of use and perceived useful of the approach according to its proposed guidelines. The results shown that both approaches were perceived to be easy to use and useful for Brazilian and Spanish subjects. Thus, since each approach was evaluated in a different country, it was possible to strengthen the empirical studies results and get more feedback about the effectiveness and usefulness of the approaches.

1.4 Research History

This Section describes the history of this Ph.D. investigation, which officially started in 2011. Figure 1.3 outlines the major activities performed up to date. It consists of one continuum that represents the current Ph.D. investigation. These are detailed next.

- **Ph.D. course starts.** In the year 2011 this Ph.D. research started. The first year (from March/2011 to February/2012) was dedicated to take the regular courses, mandatory in our Graduate program. Besides taking classes, it was possible to run two industrial exploratory empirical studies within companies in Salvador, Bahia, Brazil.
- **Scope definition.** As soon as the regular courses ended, we started discussing about the intended goals to this investigation. By analysing the data from the two empirical studies, observing practices available in the literature, and rounds of discussion with internal and external members of our research group, we could establish a research agenda, which included the major research investigation goal: to provide a better support for SPL evolution.
- **Internship.** Over six months (from October/2012 to March/2013), this research was carried out in the Software Engineering and Information Systems (ISSI) research group, at Universidad Politècnica de Valencia, in Spain under the supervision of Prof. Dr. Silvia

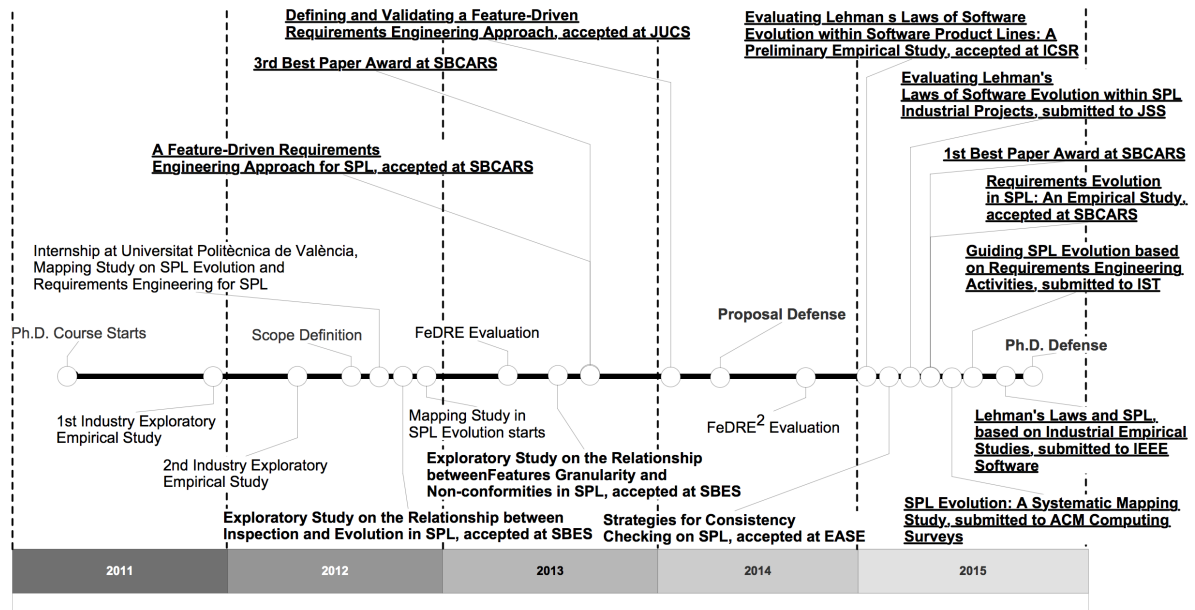


Figure 1.3: Research Timeline.

Abrahão and Prof. Dr. Emilio Insfran. This was a labor-intensive period in which it was possible to share our ideas and discuss them with a very productive research group. Such a period provided us with the opportunity to start an SPL evolution mapping study and also improve our knowledge within SPL requirements engineering, where it was defined and evaluated FeDRE approach.

- Proposal defense.** In early June of 2014, the proposal (*Feature-Driven Requirements Engineering Evolution Approach for Software Product Lines - FeDRE²*) was presented and assessed by a board committee. In addition to the formalism requested by the Graduate Program, this was the time to discuss on top of the proposed idea, and draws further steps. After the Proposal Defense, we made some improvements within the work of this Thesis. The first improvement was related to the Empirical studies evaluating the Lehman's laws of software evolution within SPL. We changed the statistical method to evaluate the formulated hypothesis from Augmented Dickey Fulley Test to KPSS test, which led us to a better statistical analysis. The second improvement was to update the SPL evolution systematic mapping study including papers published from 1996 up to 2014. The third improvement was the evaluation of FeDRE². This approach was evaluated within Brazilian and Spanish subjects.
- Ph.D. defense.** On September 10th of 2015, this Ph.D. Thesis was presented and assessed

by a board committee in fulfillment of the requirements for the degree of Philosophy Doctor in Computer Science.

- **Some results.** The bold highlighted texts in Figure 1.3 are the main results so far. The underline ones are strictly related to this Thesis. At the end of 2012, using the data collected from the private exploratory empirical study, we had an accepted paper at the Brazilian Symposium on Software Engineering (SBES) where we investigated the relationship between information from features non-conformities and data from corrective maintenance: *On the Relationship between Inspection and Evolution in Software Product Lines: An Exploratory Study* (Souza *et al.*, 2012). At the end of 2013, we had two accepted papers. The first one was an investigation of a possible correlation between feature granularity and feature non-conformity, accepted at SBES: *On the Relationship between Features Granularity and Non-conformities in Software Product Lines: An Exploratory Study* (Souza *et al.*, 2013). The second one was a paper accepted at Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS): *A Feature-Driven Requirements Engineering Approach for Software Product Lines* (Oliveira *et al.*, 2013). This paper is the approach for specifying SPL requirements in a systematic way, defined in partnership with the ISSI group from Spain and Sholom Cohen from Software Engineering Institute (SEI). This paper also received the 3rd best paper award at SBCARS'13. We improved the SBCARS paper, mainly the guidelines for specifying SPL requirements and the case study, and we had an accepted paper at Journal of Universal Computer Science (J.UCS), special issue: Software Components, Architectures and Reuse: *Defining and Validating a Feature-Driven Requirements Engineering Approach* (Oliveira *et al.*, 2014). In the beginning of 2015, the first empirical study, in a medical company, evaluating the Lehman's Laws (LL) within SPL (*Evaluating Lehman's Laws of Software Evolution within Software Product Lines: A Preliminary Empirical Study*) was accepted at the International Conference on Software Reuse (ICSR) (Oliveira *et al.*, 2015a). Next, it was performed a mapping of the existing approaches to inconsistency management within SPL: *Strategies for Consistency Checking on Software Product Lines: A Mapping Study*, published at the International Conference on Evaluation and Assessment in Software Engineering (EASE) (Santos *et al.*, 2015a). The replication of the study evaluating LL within SPL, in a financial company, entitle *Evaluating Lehman's Laws of Software Evolution within Software Product Lines Industrial Projects*, was submitted to the Journal of Systems and Software (JSS) (Oliveira *et al.*, 2015b). This study evaluated the LL in another company and compared the new results (Oliveira *et al.*, 2015b) with the

previous ones (Oliveira *et al.*, 2015a). The Feature-Driven Requirements Engineering Evolution (FeDRE²) approach was evaluated and the paper describing the approach and its empirical evaluation was accepted at the Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS), entitled: *Requirements Evolution in Software Product Lines: An Empirical Study* (Oliveira and Almeida, 2015b). This paper also received the 1st best paper award at SBCARS'15. Next, the SPL evolution systematic mapping (*Software Product Lines Evolution: A Systematic Mapping Study*), covering papers from 1996 up to the end of 2014, was submitted to the ACM Computing Surveys Journal (Oliveira *et al.*, 2015d). To sum up the work performed in this Thesis, a paper called *Guiding Software Product Line Evolution Based on Requirements Engineering Activities* was submitted to the Information and Software Technology (IST) Journal (Oliveira and Almeida, 2015a). Finally, an overview of the studies evaluating Lehman's Laws (*Lehman's Laws of Software Evolution and Software Product Lines: Empirical Studies*) was submitted to IEEE Software (Oliveira *et al.*, 2015c).

1.5 Contributions

The main contributions of this work are following listed:

- **SPL Industry Empirical Studies.** We performed a first exploratory study to understand the evolution in a private SPL. The same study was replicated in another private SPL. Within the results of these studies, we identified that guidelines should exist to handle the evolution of the SPL artifacts.
- **Body of knowledge about SPL evolution.** The number of studies in the SPL evolution field is increasing year by year. We performed an SPL evolution mapping study aiming at providing the state-of-the-art evidence, that can be used in a range of investigations in the field.
- **SPL Requirement Engineering Field.** SPL requirements engineering field lacked of a systematic way for specifying requirements. We presented an approach, called Feature-Driven Requirements Engineering (FeDRE), for specifying SPL requirements, based on the feature model, in a systematic way through guidelines.
- **Evolution of SPL Requirements.** SPL requirements engineering field also lacks of a systematic way for evolving requirements. Thus, it was proposed an approach, called

Feature-Driven Requirements Engineering Evolution (FeDRE²), for evolving systematically SPL requirements, based on the feature model, through the use of guidelines.

- **Empirical Studies in Software Engineering.** In order to evaluate FeDRE and FeDRE² approaches, and improve the body of knowledge in empirical software engineering, we performed empirical studies in Brazil with a replication in Spain.

1.6 Out of Scope

It is out of scope of this Thesis, the following topics:

- *evolution of other activities in Domain and Application engineering:* how the following SPL domain activities (Architecture, Implementation, Test) and the SPL application activities (Requirement, Architecture, Implementation, Test) deal with evolution over the time;
- *evolution of other types of variability, besides the ones supported by FeDRE:* since FeDRE² approach is based on the FeDRE approach, the evolution of other types of variability, such as, variability in the use case steps and cross-cutting variability (using parameters in use case), is out of scope of this Thesis;
- *dynamic evolution:* changes within an SPL can be *static* (the SPL does not need to be running during the evolution) or *dynamic* (the SPL is running during the evolution and autonomously drives changes to itself). The latter is out of the scope of this Thesis;
- *implementation:* This Thesis deals only with the specification and evolution of SPL requirements. It does not deal with their implementation within the source code.
- *tool support:* the RiSE Labs group ¹ has a tool for supporting the development of an SPL, called SPLICE ². However, it is out of scope of this Thesis the extension of the SPLICE tool to support the specification and evolution of SPL requirements;
- *requirements heterogeneity:* this Thesis deals with requirements represented as use cases (textual and diagrams). It is out of scope of this Thesis how other representations of requirements such as, goals, and so on, evolve over time.

¹<http://www.rise.com.br/riselabs/>

²<http://www.rise.com.br/splice>

1.7 Organization of the Thesis

The thesis is structured into five parts plus appendices. Figure 1.4 shows a schematic overview of the thesis structure. Apart from the present Introduction Part I, the remainder can be outlined in the following way:

- **Part II - Background.** This part, represented by Chapter 2, provides background concepts on the topics involved in this investigation, namely software product lines, software product lines evolution, software product lines requirements engineering, and evolution of software product lines requirements.

Software Product Lines essential activities and variability management are presented in Section 2.1. Section 2.2 shows how SPL can deal with its evolution by presenting the forces of change in SPL and how the evolution is propagated among the essential activities of the SPL. The Software Product Lines Requirements Engineering (Section 2.3) shows the SPL RE concepts focusing on the activities performed by the SPL RE and presents its risks and challenges. Finally, Section 2.4 introduces the different contexts that can influence the SPL requirements evolution and also some gaps for future research.

- **Part III - Understanding Software Product Lines Evolution.** This part, divided in two Chapters, shows the studies performed in this investigation to understand the evolution within SPL.

Chapter 3 presents the empirical studies performed in two SPL industry projects, where the evolution of the SPL common, variable and product-specific assets were studied and some findings were identified. Section 3.3.2 shows the first empirical study performed in a Brazilian company which develops software for the medical domain and Section 3.3.5 presents the second empirical study performed in a Brazilian company which develops software for the financial domain. These studies revealed a decrease of quality in the artifacts over the SPL evolution process. Moreover, we identified that guidelines for evolving such artifacts should exist to make the evolution process more effective.

The next Chapter (Chapter 4) shows the mapping study in SPL evolution in order to understand and propose improvements to the area. The focus of this study is to reveal approaches that deal with SPL evolution and reveal gaps for future research.

- **Part IV - Guiding Software Product Lines Evolution based on Requirements Engineering Activities.** This part, composed of two Chapters, introduces the proposed approaches for specifying and evolving SPL requirements and their evaluations.

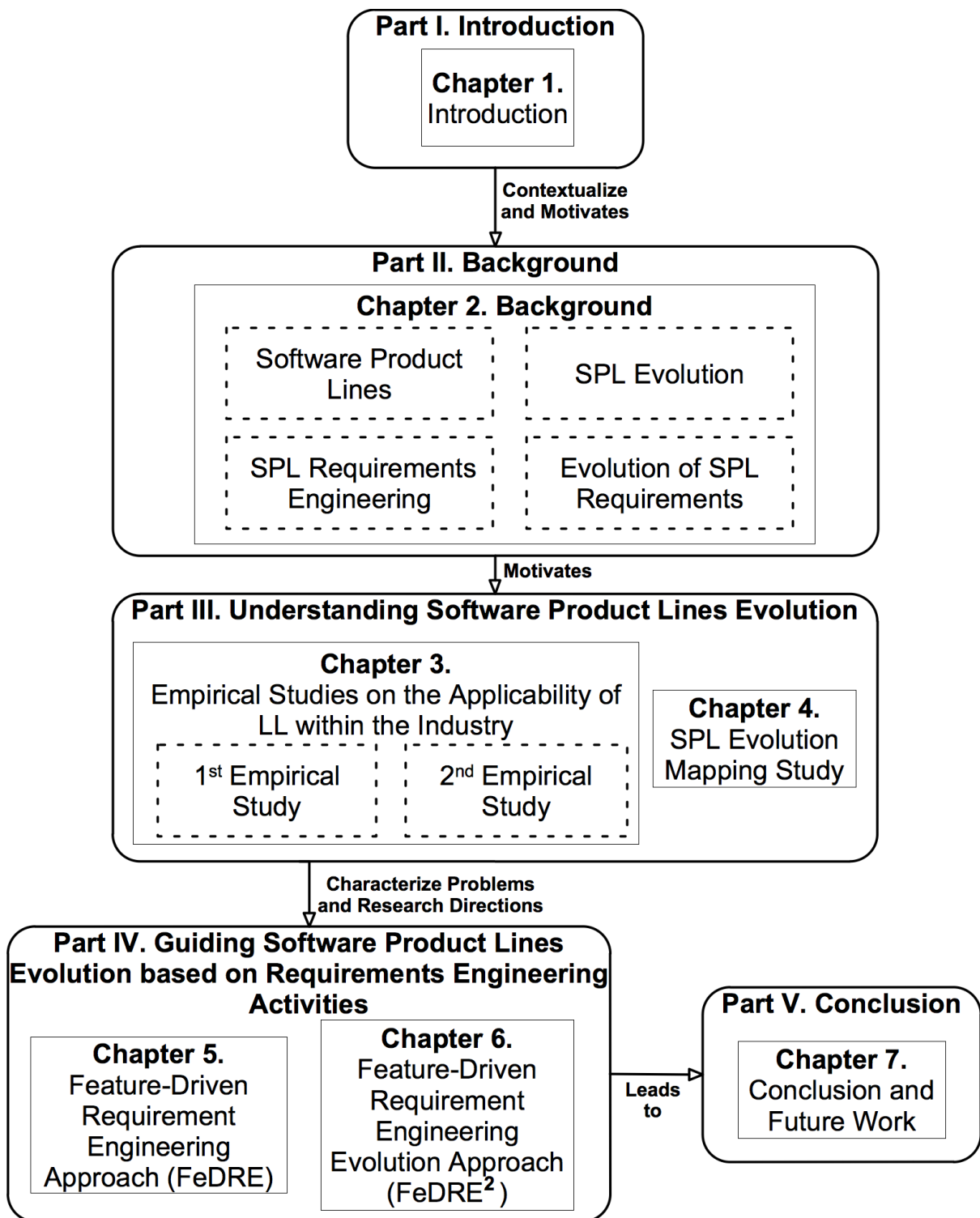


Figure 1.4: Ph.D. Thesis Organization.

The Feature-Driven Requirements Engineering (FeDRE) Approach is presented in Chapter 5. FeDRE is an approach to help in the specification of SPL domain requirements.

Chapter 6 shows the Feature-Driven Requirements Engineering Evolution (FeDRE²) Approach, which helps in the evolution of SPL domain requirements.

- **Part V - Conclusion and Future Work.** It presents conclusions and future work (Chapter 7). This Chapter presents some conclusions based on the previous performed studies and also shows the next steps to further enhance the research in this field.

PART II

Background

2

Background

In early 1967 there was an increasing importance and impact of software systems in many activities of society. In addition, there was a general belief that available techniques to build software should be less *ad-hoc*, and instead, they should be based on theoretical foundations, as the established disciplines of engineering. These were the main driving factors for organizing the first conference on Software Engineering in 1968 (Naur and Randell, 1969). The goal of this conference was “*the establishment and use of engineering principles in order to obtain reliable, efficient and economically viable software*”. Among the many activities of software engineering, this conference discussed about software mass customization (principle for Software Product Lines - SPL, Section 2.1), software maintenance (Section 2.2), user requirements (Section 2.3), and evolution of requirements (Section 2.4). These software development activities (which turned into formal software engineering fields later on) are discussed in this Chapter in the context of Software Product Lines.

2.1 Software Product Lines

The way that goods are produced has changed significantly in over the time. Previously, goods were handcrafted for individual customers (Pohl *et al.*, 2005), although each more, the number of people who could afford to buy several kinds of products have increased.

In the domain of vehicles this led to Henry Ford’s invention of the mass production (product line), which enabled production for a mass market cheaper than individual product creation on a handcrafted basis. The same idea was made also by Boeing, Dell, and even McDonald’s (Northrop, 2002).

Customers were satisfied with standardized mass products for while (Pohl *et al.*, 2005), however, not all of the people want the same kind of car. Thus, industry was challenged with the

rising interest for individual products, which was the beginning of mass customization.

Thereby, many companies started to introduce common platforms for their different types of products, by planning beforehand, which parts will be used in different product types. Thus, the use of platforms for different products led to the reduction in the production cost for a particular product kind. The systematic combination of mass customization and common platforms is the key for product lines.

Realizing the success of this combination, this concept was turned to software as Software Product Lines (SPL), which is a “*set of software-intensive systems that share a common, managed feature set, satisfying a particular market segment’s specific needs or mission and that are developed from a common set of core assets in a prescribed way*” (Clements and Northrop, 2002). The SPL essential activities are shown in next Section.

2.1.1 Software Product Line Essential Activities

Software product lines include three essential activities, as shown in Figure 2.1: Core Asset Development (CAD), Product Development (PD) and Management (Clements and Northrop, 2002). Some authors (Pohl *et al.*, 2005) use other terms for CAD and PD, for example, Domain Engineering (DE) representing the CAD and Application Engineering (AE) representing the PD. These activities are detailed as follows.



Figure 2.1: Software Product Lines Essential Activities (Clements and Northrop, 2002).

Core Asset Development (Domain Engineering). This activity focus on establishing a production capability for the products (Clements and Northrop, 2002). It is also known as Domain Engineering and involves the creation of common assets, generic enough to fit different environments and products in the same domain. Figure 2.2 illustrates the core asset development activity along with its outputs and contextual factors.

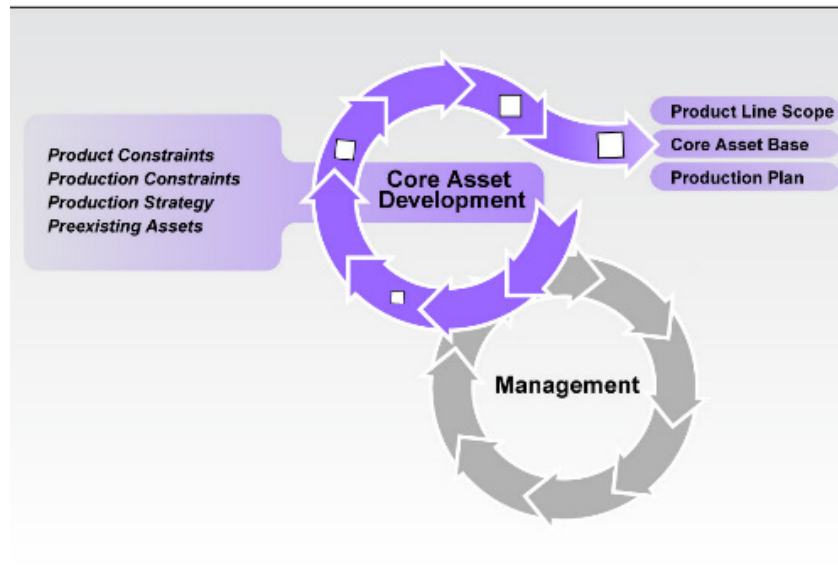


Figure 2.2: SPL Core Asset Development (Clements and Northrop, 2002).

The core asset development activity is iterative, and according to Clements and Northrop (2002) some contextual factors can impact in the way the core assets are produced. Some contextual factors can be listed as follows: *product constraints* such as commonalities, variants, and behaviors; and *production constraints*, which is how and when the product will be bring to market. These contextual factors may drive decisions about the used variability mechanisms.

Variability is one of the foundation concepts for the core asset development. Since core assets need to fulfill many product requirements at the same time, the core asset needs to have the potential to vary its behavior for the different products (variability management is discussed further in Section 2.1.2). Because of the variability inside of each core asset, it is important to exist a guideline to conduct the use of an asset inside of a product. This guideline is called *production plan* (Clements and Northrop, 2002), and should contain the production process, which is influenced by the product constraints, project details and etc, as shown in Figure 2.3.

Product Development (Application Engineering). The main goal of this activity is to create individual (customized) products by reusing the core assets. This activity is also known as Application Engineering and depends on the outputs provided by the core asset develop-

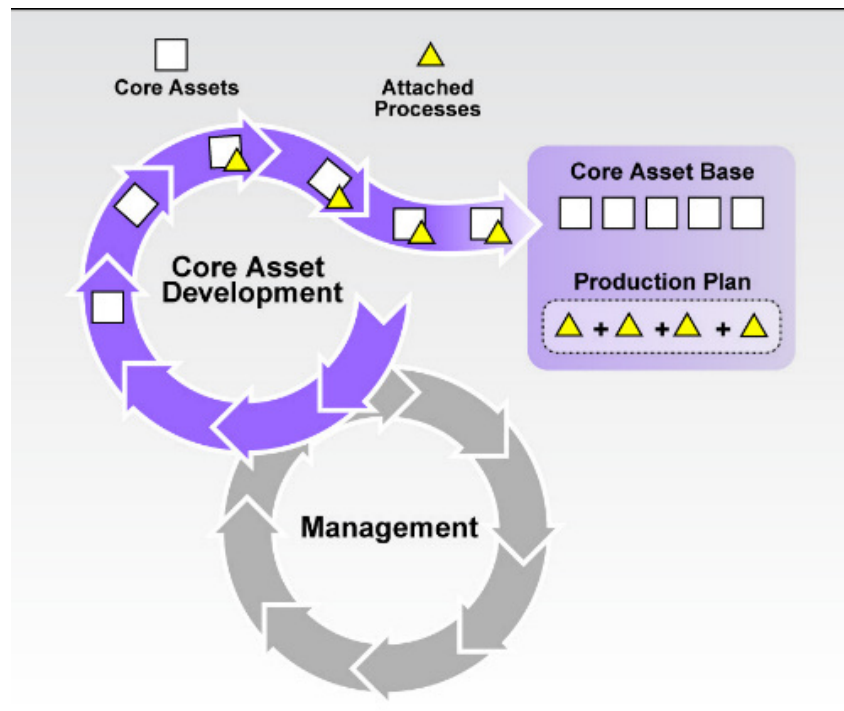


Figure 2.3: SPL Production Plan (Clements and Northrop, 2002).

ment activity (the core assets and the production plan). The relationship between Core Asset Development and Product Development activities, is illustrated in Figure 2.4.

Product engineers use the core assets, in accordance with the production plan, to produce products that meet their respective requirements. Product engineers also have an obligation to give feedback on any problem within the core assets, to avoid the SPL decay (minimizing corrective maintenance) and keep the core asset base healthy and viable for the construction of the products.

Management. This activity includes technical and organizational management. Technical management is responsible for the coordination between core asset and product development and the organizational management is responsible for the production constraints and ultimately determines the production strategy.

2.1.2 Software Product Line Variability Management

To successfully introduce software product lines concepts in a software development environment, the notion of variability is extremely important. There are common representations of variability within SPL, such as Decision Models (Weiss and Lai, 1999) and Feature Models (Kang *et al.*, 1990). Feature models are commonly used to represent the SPL variability through

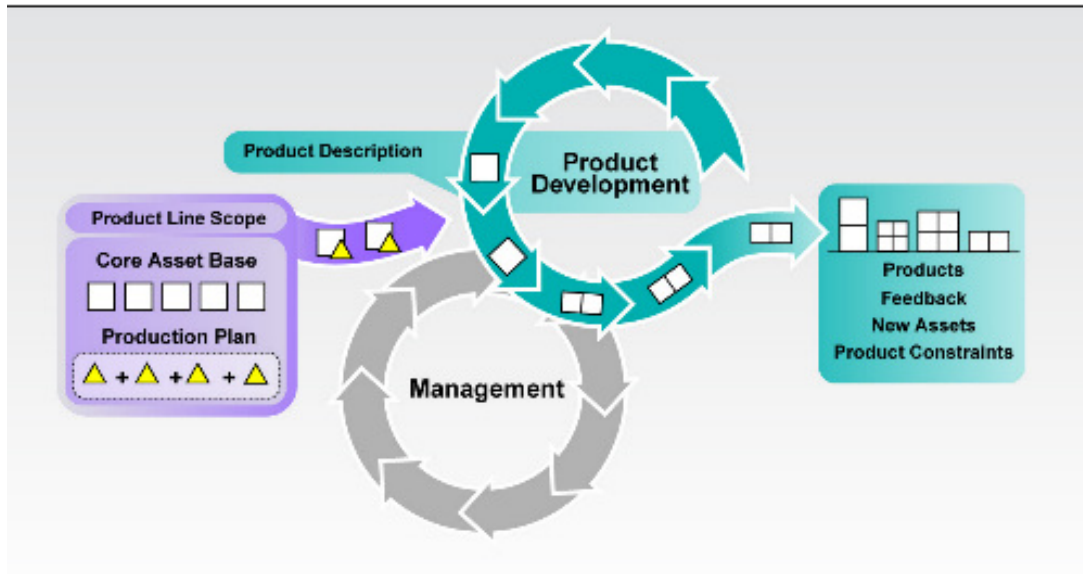


Figure 2.4: SPL Product Development (Clements and Northrop, 2002).

features. A *feature* is defined as a “prominent or distinctive user-visible aspect, quality, or characteristic of a software system or systems” (Kang et al., 1990). The SPL features are grouped in a feature model diagram to represent the SPL variability. Figure 2.5 shows a feature model diagram for the domain of cars. This feature model diagram has “concrete” features, “abstract” features, “mandatory” features, “optional” features, “or” features, and “alternative” features. “Concrete” features are features that will have code implementation. “Abstract” features will not have code implementation and they are used for grouping features in the feature model diagram. “Mandatory” features will be present in all products from the SPL. “Optional” Features may be or not present in a product configuration. “OR” Features allow the selection of one or more features from the group. “Alternative” features allow the selection of exact one feature from the group. There are also constraints among features, such as, “requires” and “excludes”. If a feature X may “requires” a feature Y, thus, when selecting the feature X, the feature Y must also be selected. On the other hand, if a feature X may “excludes” a feature Z, thus, when selecting the feature X, the feature Z must not be selected. A valid configuration of the feature diagram allows the creation of an SPL product. For example, according to the feature model diagram from Figure 2.5, a valid configuration would be the selection of the following features: CarSPL, Airbag, StabilityControlSystem->BasicSkidControl, AntilockBreakingSystem, AudioSystem->MultimediaAudioSystem->MP3Radio. By selecting these features and their associated assets (requirements, architecture, implementation, test, and so on) an SPL product can be instantiated.

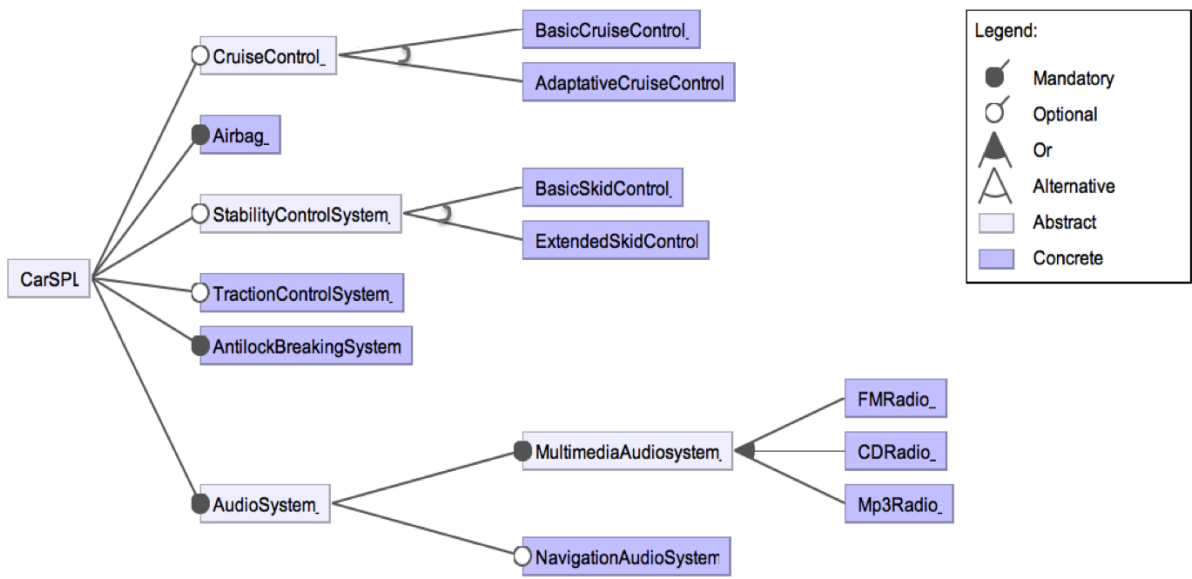


Figure 2.5: Feature Model Example.

Variability has an important role within SPL. When corrected managed, it will allow the SPL to build different products according to the selected variability. The goal of variability management inside SPL is to support the development and reuse of variable assets.

Variability management can occur in the core asset development level, with the definition of the variability inside a core asset, and also in the product development asset, by exploring the variability previously defined. Figure 2.6 shows the assets types to build SPL products. “Common” assets will be present in all SPL products. “Variable” assets will be present in some products, but not all of them. Finally, “Product Specific” assets will be part of specific products of the SPL.

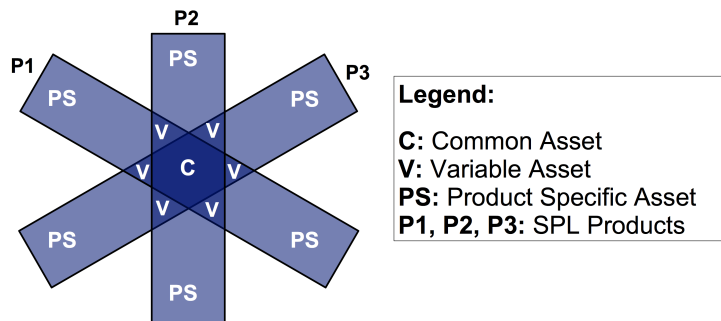


Figure 2.6: SPL Assets Types.

The variability identification observes *what vary* (the variability subject, that could be a

variable item of the real world or a variable property in such an item), *why does it vary* (the drivers of the variability need, such as stakeholder needs, technical reasons, market pressures, etc.), and *how does it vary* (the possibilities of variation, also known as variability objects).

Inside variability management, there are also the concepts of variability points and variants. The variation point is the representation of a variability subject within the core assets, enriched by contextual information. The variant is the representation of the variability object within the core assets. These two concepts are the basis for the variability definition inside an SPL (Pohl *et al.*, 2005).

The variability, according to (Krueger, 2002a), can be classified as: *variability in time*, which is the existence of different version of the same asset (or asset item) at different moments, also present in single-systems development, and are normally handled by traditional Software Configuration Management (SCM) activities; and *variability in space*, which is the existence of the same asset (or asset item) in different shapes, which is the variability for the asset.

2.2 Software Product Lines Evolution

Due to changes within the environment and users' needs, software product lines are continuously evolving, thus, its evolution should be managed properly to achieve all benefits from this approach.

Evolution in a product line is more complicated than single system because of the fact that an asset, can be shared among several products, and any change in this asset may affect all of the related products (McGregor, 2003; Botterweck and Pleuss, 2014). This makes evolution management in SPL more challenging than in traditional single software development (Pussinen, 2002).

2.2.1 Forces for Change

As in single-system development, software product lines are also subject of forces for change. Those forces may come from different contexts. According to (McGregor, 2003), the forces for change can be classified as external forces and internal forces.

External forces are one of the forces for changing in the product line organization. Some external forces are described next:

- *Potential new Competitors*. Potential competitors entering the market might force a change in the business strategies in the organization. Such a change could cause consequent changes in the product line strategy, the architecture, and related assets.

-
- *Buyers*. They might force change by demanding the latest available technology on the products they buy.
 - *Suppliers*. They might force a change by discontinuing or evolving assets they provide to the SPL.

The interactions identified in Figure 2.1 among the three essential activities result in internal forces for evolution. Some possible internal forces are described next, separated by each SPL essential activity.

- *Core Asset Development (Domain Engineering)*. The Core Asset development evolution forces the evolution on product development by providing new versions of assets and additional variants. The more frequent these releases are, the more product development resources will be consumed in order to adapt to these new versions. On the other hand, if the core asset versions wait too long to be released, it may allow the product teams to “clone and own” the assets and adapt it to their needs, breaking the flow of the SPL essential activities and making harder future evolutions for the SPL.
- *Product Development (Application Engineering)*. Product development may evolve by providing change requests to the existing assets. Besides the change requests, the product teams may discover defects and bugs in the assets, and request their fix. Product development also evolves by requesting a product specific asset to be incorporated in the core asset base, becoming a core asset reusable by other products, or it can also evolve by requesting an change within product-specific assets, without affecting the core asset and other products.
- *Management*. It drives the evolution in the core asset development by updating and adjusting the production plan for the product line. Core asset development responds to these evolutions by updating the existing core assets or creating new ones. Management also drives the evolution in the product development, by modifying the production plan and the product line scope.

Some of the examples of internal forces for changes, may result in the propagation of the changes (evolution). The evolution propagation is described next.

2.2.2 Evolution Propagation

In traditional single-system development, the changes performed into an artifact may have impacts on other artifacts of the system. In the software product line context, where changes to

an artifact may impact dozens or hundreds of assets and products, the impacts of evolution are even bigger.

In certain situations (such as the internal forces for changes described in section 2.2.1) one change in a SPL development level (core asset development or product development) may have effects on the other one. Because of that, the changes performed to an asset or a product have to be propagated to the other development level, to keep the products or assets compliant with each other. Thus, the propagation of changes (evolution propagation) may occur in different directions (Atkinson *et al.*, 2002; Botterweck and Pleuss, 2014):

- *Internal Propagation (Core Asset Development / Domain Engineering)*. When a core asset change only affects other core assets. Thus, the change propagation is performed internally in the core asset development level.
- *Internal Propagation (Product Development / Application Engineering)*. When a product change only has effects on a specific product. Thus, the change propagation is performed internally in the product development level.
- *Propagation from Core Asset Development to Product Development*. Every new core asset release (comprising a set of changes) can be propagated to the product development level, in order to keep the products up-to-date with the latest versions of core assets (with fixed defects, improvements, etc). In this context, the changes are propagated from the core asset development to the product development level.
- *Propagation from Product Development to Core Asset Development*. Whenever a core asset is modified in the product development level, that change may be propagated back to the core asset development level. Also, when a product specific asset should become a core asset, the product specific asset is propagated to the core asset development level, and integrates the core asset base.

The changes performed within SPL are firstly represented by requirements. Thus, the next Section presents how SPL requirements engineering is performed.

2.3 Software Product Lines Requirements Engineering

Requirements are typical assets in SPL. They are specified in reusable models, in which commonalities and variabilities are documented explicitly. Thus, these requirements can be instantiated and adapted to derive the requirements for an individual product (Cheng and Atlee,

2007). During product derivation, for each variant asset, it is decided whether the asset is (or is not) supported by the product to be built. When a domain requirement is instantiated, it can become a concrete product requirement. Thus, new products in the SPL will be much simpler to specify, because the requirements are reused and tailored (Clements and Northrop, 2002).

Deciding which products to build depends on business goals, market trends, technological feasibility, and so on. On the other hand, there are many sources of information to be considered and many trade-offs to be made. The SPL requirements must be general enough to support reasoning about the scope of the SPL, predicting future changes in requirements and anticipated SPL growth.

In practice, establishing the requirements for an SPL is an iterative and incremental effort, covering multiple requirements sources with many feedback loops and validation activities (Chastek *et al.*, 2001). Thus, *Requirement Engineering (RE)* in SPL has an additional cost. Many SPL requirements are complex, interlinked, and divided into common, variable and product-specific requirements (Birk *et al.*, 2003; Oliveira *et al.*, 2014). Regarding to single systems, RE for SPL has some differences, such as (Clements and Northrop, 2002; Pohl *et al.*, 2005; Thurimella and Bruegge, 2007):

- *Elicitation* captures anticipated variations over the foreseeable life-cycle of the SPL. RE must anticipate prospective changes in requirements, such as laws, standards, technology changes, and market needs for future products. Thus, its sources of information are probably larger than for single-system requirements elicitation.
- *Analysis* identifies variations and commonalities, and discovers opportunity for reuse.
- *Negotiation* solves conflicts not only from a logical viewpoint, but also taking into consideration economical and market issues. The SPL requirements may require sophisticated analysis and intense negotiation to agree on both common requirements and variation points that are acceptable for all the systems.
- *Specification* documents a SPL set of requirements. Notations are used to represent the product line variabilities and enable the product instantiation.
- *Verification* checks if the SPL requirements can be instantiated for the products, ensuring the reusability of the requirements.
- *Management* must provide a systematic mechanism for proposing changes, evaluating how the proposed changes will impact the SPL, specifically its core asset base. Evolution

can affect the reuse and customization, therefore, appropriate mechanisms must be used to manage the variabilities.

In SPL, RE also has influence of several stakeholders that participate of the SPL. Identifying stakeholders that directly influence the RE is essential to define the requirements negotiation participants. They are responsible for resolving conflicts and providing information.

Each stakeholder plays a role with respect to the SPL. Many of the stakeholders that help to define the requirements also use them. These users have different expectations of the outputs of SPL analysis. Some may simply want to confirm that their interests have been represented (*e.g.*, marketers, domain expert and analyst domain). Others (*e.g.*, architects and developers) may want to describe proposed functional and non-functional capabilities, and their commonality and variability across the SPL, thus, those decisions about architectural solutions and asset construction should be taken into account (Chastek *et al.*, 2001).

Several approaches to deal with the definition and specification of functional requirements in SPL development have been proposed over the last few years. Some approaches specify the SPL requirements through features and use cases (Griss *et al.*, 1998; Bayer *et al.*, 1999a; Moon *et al.*, 2005; Eriksson *et al.*, 2005; Bonifácio and Borba, 2009; Alférez *et al.*, 2011; Mussbacher *et al.*, 2012; Shaker *et al.*, 2012). An SPL functional requirement represented as an use case has at least the following fields: *identifier*, *name*, *description*, *associated feature(s)*, *pre and post-Conditions*, and *the Main Success Scenario*, as shown in Table 2.1. It may also has *alternative scenarios*, *includes/extends relationships*, and so on. The feature associated to the use case handles the variability within the SPL.

Table 2.1: SPL Use Case Example.

*ID:	Use case identifier		
*Name:	Use case name		
*Description:	Use case description		
Associated feature:	Feature associated to the use case	Actor(s) [0..]:	Actor associated to the use case
*Pre-condition:	Use case pre-condition	*Post-condition:	Use case post-condition
*Main Success Scenario			
Step	Actor Action	Blackbox System Response	
Step represented by a number	Actor action	System response	

* Mandatory Field

However, the approaches for specifying SPL functional requirements do not propose guidelines, showing step by step how the specification should be done. This lack of guidelines may led to some challenges and risks.

2.3.1 Risks and Challenges

A key RE challenge for SPL development includes strategic and effective techniques for analyzing domains, identifying opportunities for SPL, and identifying the commonalities and variabilities of an SPL (Cheng and Atlee, 2007). Another challenge related to RE is that the applicability of more systematic techniques and tools is limited, partly because such techniques are not yet designed to cope with SPL development's inherent complexities (Birk *et al.*, 2003).

Regarding to the risks associated with RE for SPL, the major risk is failure to capture the right requirements, and their variabilities, over the life of the SPL (Clements and Northrop, 2002). Documenting the wrong or inappropriate requirements, failing to keep the requirements up-to-date, or failing to document the requirements at all, may affects the subsequent activities (architecture, implementation, tests, and so on). They will be unable to produce systems that satisfy the customers and fulfill the market expectations. Moreover, inappropriate requirements can result from the following (Clements and Northrop, 2002):

- *Failure in the communication between core assets requirements development and product requirements development.* The core asset builders need to know the requirements they must build, while the product-specific software builders must know what is expected of them. The lack of communication between these two development stages may led to inconsistent requirements or even unnecessary variabilities in the requirements.
- *Insufficient generality.* Insufficient generality in the requirements leads to a design that is too fragile to deal with the change actually experienced over the life-cycle of the SPL.
- *Excessive generality.* Excessive generality on requirements lead to excessive effort in producing both core assets (to provide that generality) and specific products (which must turn that generality into a specific instantiation).
- *Wrong variation points.* Incorrect determination of the variation points results in inflexible products and the inability to respond rapidly to customer needs and market shifts.
- *Failure to account for qualities other than behavior.* SPL requirements (and software requirements in general) should capture requirements for quality attributes such as performance, reliability, and security.

In order to minimize the risks within SPL requirements engineering, the evolution of SPL requirements should be addressed in an effective way considering the SPL context.

2.4 Evolution of Software Product Lines Requirements

In general, the SPL requirements evolution decisions can be justified by the SPL context. There are different contexts that can influence the SPL requirements evolution decisions, such as:

- **Starting situation:** An SPL can start from the scratch, it can be introduced while some products are already under development, or the new requirements can be re-engineered from legacy systems (Bayer *et al.*, 1999b). The latter influences the evolution of the requirements specification from single system requirements to SPL requirements.
- **Market orientation:** An SPL can be developed for a specific market segment without a concrete customer in mind, or for individual customer projects (Birk *et al.*, 2003). This situation influences the identification of the stakeholders involved in the SPL requirements evolution. For example, whether the SPL is oriented for a market segment, potential information sources for the SPL requirements evolution can be marketers. However, whether the SPL is oriented for a specific customer, this customer is a potential stakeholder for the SPL requirements evolution.
- **Domain maturity:** The domain may exist for quite some time and it is well understood, or the domain may be new. This influences the problem understanding, so it can be essential for evolution decisions in relation to the detail level of the SPL requirements specification.
- **Business constraints:** Business constraints (*e.g.*, time to market and resources) can influence the SPL requirements evolution.
- **Organizational context:** The organizational structure and behavior can influence the SPL requirements evolution, such as team size, organizational structure and available stakeholders.
- **Geographical context:** The geographical separation can also influence the SPL requirements evolution, since different means for communication and coordination are used, depending on the distance among collaborators (Fricker and Stoiber, 2008).
- **Domain stability:** Domain may not be expected to change in the near future, or the domain may change with frequency. This situation must be analyzed to negotiate requirements and identify best strategies for changes management. Whether the domain is not stable, it requires a more systematic SPL evolution management.

According to previous studies, there is a lack of approaches that deal with SPL requirements evolution in a systematic way (Alves *et al.*, 2010; Oliveira and Almeida, 2015b). Thus, the definition and use of guidelines for performing the evolution of the requirements in SPL may improve the effectiveness of the SPL, since requirements are the basis of the SPL development. The guidelines should deal with the most common evolutions in SPL through well defined steps, keeping the traceability among the artifacts over the evolution process.

2.5 Chapter Summary

Software product lines development aims at reducing costs, reducing time to market, improving the software quality, and improving the reuse. However, SPL are also under pressure of evolution due to external and internal forces to change the requirements. New software requirements are always part of requests due to the fact that the software must adapt to its environment and user's needs to keep the user's satisfaction, otherwise, the software is discontinued. Thus, in order to cope with new SPL requirements and improve the SPL evolution process, there is a need for understanding the SPL evolution area and proposing new approaches for dealing with the SPL requirements, since they are the basis of the SPL development and may affect the whole SPL.

Next Chapter presents the studies performed to understand the SPL evolution field, where we performed two empirical studies within the industry and a systematic mapping study.

PART III

Understanding Software Product Lines Evolution

3

Empirical Studies on the Application of Lehman's Laws within the Industry

The evolution of SPL assets is an important activity to keep the SPL users' satisfaction and also avoid the SPL to break down. This Chapter starts a series of studies performed to understand the SPL evolution. The first study (Oliveira *et al.*, 2015a) presents a first empirical study performed in an industrial SPL project in Salvador, Bahia, Brazil, which develops software for the medical domain. The second study (Oliveira *et al.*, 2015b) presents an empirical evaluation performed in another industrial SPL project in Salvador, Bahia, Brazil, which develops software for the financial domain. The evolution of the different SPL assets (common, variable, and product-specific) were evaluated within both studies and some findings were identified.

The remainder of this Chapter is organized as follows: Section 3.1 introduces the background concepts and the objective of the studies. Section 3.2 discusses related work and uses it as context to position this work. Section 3.3 describes both empirical investigations, including a comparison between their results. Section 3.4 presents the threats to validity of our studies. Next, it is discussed the key findings and contributions to the SPL community in Section 3.5. Finally, Section 3.6 presents the Chapter summary.

3.1 Introduction

Software evolution is a very important activity where the software must have the ability to adapt according to the environment or user needs, to keep its satisfactory performance, (Mens and Demeyer, 2008) given that if a system does not support changes, it will gradually lapse into uselessness (Lehman, 1980).

Back in the 1970s, Meir Lehman started to formulate his laws of software evolution, after

Table 3.1: Lehman’s Laws of Software Evolution (Herraiz *et al.*, 2013).

Formulation Year	Software Evolution Laws	Description
Evolution of Software System Characteristics (ESSC)		
(1974)	Continuing change	An E-type system must be continually adapted, or else it becomes progressively less satisfactory in use.
(1974)	Increasing complexity	As an E-type is changed, its complexity increases and becomes more difficult to evolve unless work is done to maintain or reduce the complexity.
(1980)	Continuing growth	The functional capability of E-type systems must be continually enhanced to maintain user satisfaction over system lifetime.
(1996)	Declining quality	Unless rigorously adapted and evolved to take into account changes in the operational environment, the quality of an E-type system will appear to be declining.
Organizational/Economic Resource Constraints (OERC)		
(1980)	Conservation of organizational stability	The work rate of an organization evolving an E-type software system tends to be constant over the operational lifetime of that system or phases of that lifetime.
(1980)	Conservation of familiarity	In general, the incremental growth (growth rate trend) of E-type systems is constrained by the need to maintain familiarity.
Meta-Laws (ML)		
(1974)	Self regulation	Global E-type system evolution is feedback regulated.
(1996)	Feedback System	E-type evolution processes are multilevel, multiloop, multiagent feedback systems.

realizing the need for software systems to evolve. These laws, shown in Table 3.1, stressed that a system needed to evolve due to its requirement to *operate in or address a problem or activity in the real world*, what Lehman called *E-type Software*.

According to Barry *et al.* (2007), these laws can be ordered into three broad categories: (i) laws about the *Evolution of Software System Characteristics (ESSC)*; (ii) laws referring to *Organizational/Economic Resource Constraints (OERC)* on software evolution; and (iii) *Meta-Laws (ML)* of software evolution. However, these laws were evaluated in the context of single systems.

In SPLs, the evolution needs a special attention since sources of SPL changes can be targeted to the entire product line (affecting common assets), targeted to some products (affecting variable assets), or targeted to an individual product (affecting product-specific assets) (Svahnberg and Bosch, 1999; Ajila and Kaba, 2004; Bailetti *et al.*, 2004).

Thus, the objective of this study is to examine whether Lehman's Laws (LL) are reflected in the development of SPLs, where common, variable and product specific assets are built. The hypotheses that we put forward is that there is a relationship between LL of Software Evolution and the software evolution in SPL environments. In this context, in order to understand whether there is a relationship between the LL of software evolution and the SPL evolution process, it was carried out two empirical investigations in industrial SPL projects. As a preliminary study, it was selected the first and the second group of laws (*Evolution of Software System Characteristics - ESSC*, which includes the Continuing change, Continuing growth, Increasing complexity, and Declining quality laws; and *Organizational/Economic Resource Constraints - OERC*, which includes the Conservation of Familiarity and the Conservation of Organizational Stability laws) to perform the evaluation. So far, the *Meta-Laws - ML (Self Regulation and Feedback System laws)* were not evaluated in both industrial projects. Thus, it was evaluated each one of the laws from the ESSC and OERC groups for common, variable and product-specific assets in the context of both industrial SPL projects. We consider that the variation points are implicit represented by common and variable assets from these SPLs. To the best of our knowledge, this is the first study stating that most of evaluated LL of software evolution can be applied in the context of SPL.

3.2 Related Work

Since the publication of Lehman's work on software changes, other researchers have investigated his laws within the context of open source and industrial projects.

The first empirical work on software changes was carried out by [Lehman \(1980\)](#), in which he used a large-scale commercial system, the IBM OS/360, to validate his laws. From this work, two well know Laws of Software Evolution could be summarized as follows ([Gupta et al., 2010](#)): Law of continuing change (a system that is being used undergoes continuing change) and Law of increasing complexity (a computer program that is changed becomes less and less structured). After the publication of Lehman's eight laws of software evolution ([Lehman et al., 1997](#)), other researchers started to check their validity within open source and industrial software.

[Godfrey and Tu \(2000\)](#) explored the evolution of the Linux kernel both at the system level and within the major subsystems and found out that the Linux has been growing in a super-linear rate over the years. However, as will be detailed later, within the context of this study it was found a different behavior. The complexity within the assets has grown over the years and the quality has decreased. It is important to notice that the number of maintainers in a private context

is smaller compared to maintainers of the Linux kernel and also the time-to-market pressure in a private context can influence the overall software product quality.

Barry *et al.* (2007) also investigated Lehman's Laws (LL); however within the context of industrial projects. They proposed some metrics as dependent variables (*number of activities; module count; cyclomatics per module, operands per module, and calls per module; number of corrections per module; percentage growth in module count; number of activities per developer*), which were also related to six LL (the *self-regulation* and the *feedback system* laws were not investigated in this study). In their study, four laws were supported (continuing change, continuing growth, declining quality, and conservation of familiarity laws) and two were not supported (increasing complexity and conservation of organizational stability). In this Thesis, some of these metrics, proposed by Barry *et al.* (2007), were adapted (as shown in the next section) to support and evaluate the LL in industrial SPL projects.

Xie *et al.* (2009) also investigated LL by studying 7 open source applications written in C and several laws were confirmed in their experiment. Their analysis covered 653 releases in total and sum 69 years of software evolution including the 7 applications. According to the authors, the definition of the *increasing complexity* and *declining quality* laws may lead to misinterpretations, and the laws could be supported or rejected, depending on the interpretation of the law definition. In this Thesis, to avoid this misinterpretation, it is considered that the increasing of complexity and the declining of quality must happen to support these laws.

Israeli and Feitelson (2010) examined LL within the context of the Linux kernel. They selected the Linux kernel because of its 14 years data recording history about the system's evolution, which includes 810 versions, and also because within the Linux kernel they had access to all the source code. Thus, they were able to evaluate all of the laws within the Linux kernel and realized that the Linux kernel is an example of a *perpetual development*, where the development is performed in collaboration with its users. Hence, most of the laws were supported in their evaluation and only two out of the eight laws were not supported (*i.e.*, *self-regulation* and *feedback system*).

Lotufo *et al.* (2010) studied the evolution of the Linux kernel variability model. This model is responsible for describing features and configurations from the Linux kernel. They found that the feature model grows together with the code. Besides the growth of the number of features, the complexity still remains the same. Most of the evolution activity is related to add new features. Their results showed that evolving large variability models is feasible and does not necessarily deteriorate the quality of the model. Some might claim that the Linux kernel is an SPL, but since it does not follow the SPL development paradigm we did not consider it as an SPL.

Herraiz *et al.* (2013) analyzed the most relevant publications dealing with LL over a period of more than 40 years. They started with studies that led to the formulation of the laws and finished with studies which elaborate questions or confirm their validity. According to them, not only the software systems need to evolve, but also their environment and the formulated laws.

Gonzalez-Barahona *et al.* (2014) studied the evolution of a long lived FLOSS¹ software project, called glib, based on information of around 20 years in the repository. They observed which information could be extracted from the repository and then evaluated the laws according to the available retrieved information (*i.e.*, number of commits, number of files per commit, number of changes, lines added and removed, Lines of Code (LOC), Source Lines of Code (SLOC)). They used these retrieved information to evaluate most of the laws, however, others (self-regulation, declining quality, and feedback system) could not be evaluated with the available data. The findings for this study were: the continuing change law was completely supported; the increasing complexity law was supported only in part and during some periods; the conservation of organizational stability law was supported at least to a certain extent; the conservation of familiarity law was not supported during most of the life of glibc and; the continuing growth law was not supported during the last phase of the project.

Godfrey and German (2014) performed some observations on modern software evolution and LL. They discuss how LL may need to evolve to accommodate new trends (*i.e.*, agile development, cloud-based services, powerful run-time environments). According to them, new metrics and new empirical models are necessary to deal with new approaches of software development and reuse. Thus, LL are “*beholden*” to Lehman’s first law (continuing change), in which we need to continually adapt them or they will become less useful (Godfrey and German, 2014).

The ML group of laws (*Self Regulation* and the *Feedback System* laws) were not evaluated in this empirical study. Barry *et al.* (2007) and Gonzalez-Barahona *et al.* (2014) also did not evaluate them. According to Barry *et al.* (2007), it is difficult to state which empirical model could be used to support or reject these laws. Gonzalez-Barahona *et al.* (2014) said that these ML require “feedback mechanisms” to be evaluated, which are key to their formulation. Thus, according to their available data, they could not evaluate them. We faced the same challenge within this empirical study. Thus, we could not find an empirical model neither available feedback data within the project to evaluate the ML.

The empirical studies performed and presented in this Chapter have a long history of data, with more than 10 years of evolution records, as many of the presented related work.

¹Free Open Source Software - <http://freeopensourcesoftware.org>

Nevertheless, they were performed with private SPLs, allowing the evaluation of the laws for the common, variable and product-specific assets.

3.3 Empirical Studies

The empirical studies presented herein focuses on investigating the relationship between LL of software evolution and the common, variable and product-specific assets, based on data from two industrial SPL projects. They were planned and executed according to [Jedlitschka et al. \(2008\)](#) and [Carver \(2010\)](#) guidelines.

The first empirical study was performed in the medical domain and the second one was performed in the financial domain, both in an industrial setting. Since both empirical studies used the same planning, it is presented first the general planning and after the details of each empirical study.

3.3.1 General Planning

The planning for both empirical studies started by defining their goals based on the GQM approach ([Basili et al., 1994](#)), as follows: the goal of each empirical study is *to analyze Lehman's Laws of Software Evolution* for the purpose of *evaluation* with respect to *its validity* from the point of view of *the researcher* in the context of *an industrial SPL project*. Based on the stated Goal, it was addressed the following research questions:

- RQ1. Is there a relationship between the *Continuing Change* law and the evolution of common, variable, and product-specific assets?
- RQ2. Is there a relationship between the *Continuing Growth* law and the evolution of common, variable, and product-specific assets?
- RQ3. Is there a relationship between the *Increasing Complexity* law and the evolution of common, variable, and product-specific assets?
- RQ4. Is there a relationship between the *Declining Quality* law and the evolution of common, variable, and product-specific assets?
- RQ5. Is there a relationship between the *Conservation of familiarity* law, and the evolution of common, variable and product-specific assets?

RQ6. Is there a relationship between the *Conservation of organizational stability* law and the evolution of common, variable, and product-specific assets?

The term relationship used in the RQs seeks for evidences of each evaluated law in the SPL common, variable, and product specific assets. In order to answer those questions, some metrics were defined. Since the SPL literature does not provide clear metrics directly associated to the laws, the metrics extracted from Barry *et al.* (2007), Xie *et al.* (2009), and Kemerer and Slaughter (1999) were used herein. Barry *et al.* defined some dependent variables and some metrics for measuring each dependent variable. Based on their work, in order to evaluate each LL of software evolution, it was adapted the relationship among laws, dependent variables and the measurements (as shown in Table 3.2), according to the available data in the private industrial environment. Moreover, instead of using the Cyclomatic Complexity (McCabe, 1976) as in Barry's work, we decided to use the LOC metric, since LOC and Cyclomatic Complexity are found to be strongly correlated (Kan, 2002).

For each one of the dependent variables, we have stated one null and one alternative hypothesis. The hypotheses for the empirical study are shown next:

H_0 : There is no growth trend in the data during the years (Stationary);

H_1 : There is a growth trend in the data during the years (Trend);

To corroborate Lehman's Laws of Software Evolution, *Continuing Change*, *Continuing Growth*, *Increasing Complexity* and *Declining Quality*, we must reject H_0 . Thus, if there is a trend of growth in the data during the years, there is evidence to support these four laws. However, for the *Conservation of Familiarity* and the *Conservation of Organizational Stability* laws, we must not reject H_0 . Thus, no growth trend should exist in the data during the years for these two laws.

In order to evaluate the hypotheses it was applied the KPSS Test (Kwiatkowski *et al.*, 1992). This test is used to test a null hypothesis for an observable time series. If the series is stationary, then we do not reject the null hypothesis. Otherwise, if the series has a trend, we reject the null hypothesis. In this study, the level of significance used was 5% (alpha-value). We could evaluate 95% of the assets (common, variable and product-specific) for all the laws using this statistical test.

It was also applied a linear regression analysis (Yan and Su, 2009) to the collected data to evaluate the variance between the assets of the industrial SPL projects. Through this variance, it was possible to understand which assets evolve more and should receive more attention. It was checked the variance ($Y = \beta_0 + \beta_1 X$) for each dependent variable and for each asset (Common

Table 3.2: Relationship among Laws, Dependent Variables and Measurement, based on [Barry et al. \(2007\)](#).

Law	Dependent Variable	Acronym	Measurement
Continuing change	Number of Activities	NA	Count of <i>corrective, adaptive</i> and <i>perfective</i> requests over the years (Barry et al., 2007)
Continuing growth	Lines Of Code	LOC	Number of lines of code of modules over the years (Xie et al., 2009)
Increasing complexity	Number of Corrections per LOC	NCLOC	Total of <i>correction</i> requests divided by LOC of modules over the years (adapted from Kemerer and Slaughter (1999))
Declining quality	Number of Corrections per Module	NCM	Total <i>correction</i> requests divided by the number of modules over the years (Barry et al., 2007)
Conservation of familiarity	Relative Growth in Module count	RGM	New modules created in the year divided by the total number of modules from the previous year (Barry et al., 2007)
Conservation of organizational stability	Number of Activities per Developer	NAD	Count of <i>corrective, adaptive</i> and <i>perfective</i> requests divided by the count of developers in the year (Barry et al., 2007)

= comm, Variable = var, Product-Specific = ps) of the SPL. Next, it is presented details of the first empirical study ([Oliveira et al., 2015a](#)).

3.3.2 First Empirical Study

The industrial SPL project, used as basis for the investigation described herein, has been conducted in partnership with a company located in Brazil, which develops for more than 10 years strategic and operational solutions for hospitals, clinics, labs and private doctor offices. This company has ~ 50 employees, of which six are SPL developers with a range of 4 to 19 years of experience in software development.

The company builds products, using the PowerBuilder integrated development environment², within the scope of four main areas (hospitals, clinics, labs and private doctor offices). Such

²<http://www.powerbuilder.eu/>

products comprise 45 modules altogether, targeting at specific functions (*e.g.*, financial, inventory control, nutritional control, home care, nursing, and medical assistance). Market trends, technical constraints and competitiveness motivated the company to migrate their products from a single-system development to an SPL approach. Within SPL, the company was able to deliver its common, variable and product-specific assets. To keep the company name confidential, it will be called *Medical Company (MC)*. During the investigation, MC allowed full access to its code and bug tracking system.

Regarding the bug tracking system, it was collected a total of 70,652 requests over 10 years, allowing an in-depth statistical data analysis. Thus, for MC we analyzed the data through the years. MC uses a bug tracking system called *Customer Interaction Center (CIC)*, which was internally developed. CIC allows MC's users to register requests for adaptations, enhancements, corrections and also requests for the creation of new modules.

All the products at MC have some assets (called modules) in common (commonalities), some variable assets (variabilities) and also some specific assets (product-specific), enabling the creation of specific products depending on the combination of the selected assets.

Figure 3.1 shows the division of modules between the areas supported by MC. Four (4) modules represent the commonalities of the MC SPL, twenty-nine (29) modules represent the variabilities of the MC SPL and, twelve (12) modules represent the product-specific assets, totaling forty-five (45) modules in the MC SPL. These modules were classified into common, variable, and product-specific according to their usage in the MC supported areas. A senior developer helped us in this classification, since the modules did not have clear boundaries among them.

Based on those modules, some of the laws could be evaluated with the records from CIC. However, other ones required the LOC of these modules. From CIC and LOC, we collected data since 1997. Nevertheless, data related to the three types of maintenance (adaptive, corrective and perfective) just started to appear in 2003.

The next subsection describes how the data were collected and grouped to allow the evaluation of the defined hypotheses.

3.3.2.1 Execution

The object of this study was the MC SPL. To collect the necessary data (from source code and the bug tracking system), we defined an approach composed of three steps. In the first step, we were able to collect data from *Customer Interaction Center (CIC)*, in the second one we collected LOC data and at the third step, MC clarified some doubts, through interviews, that

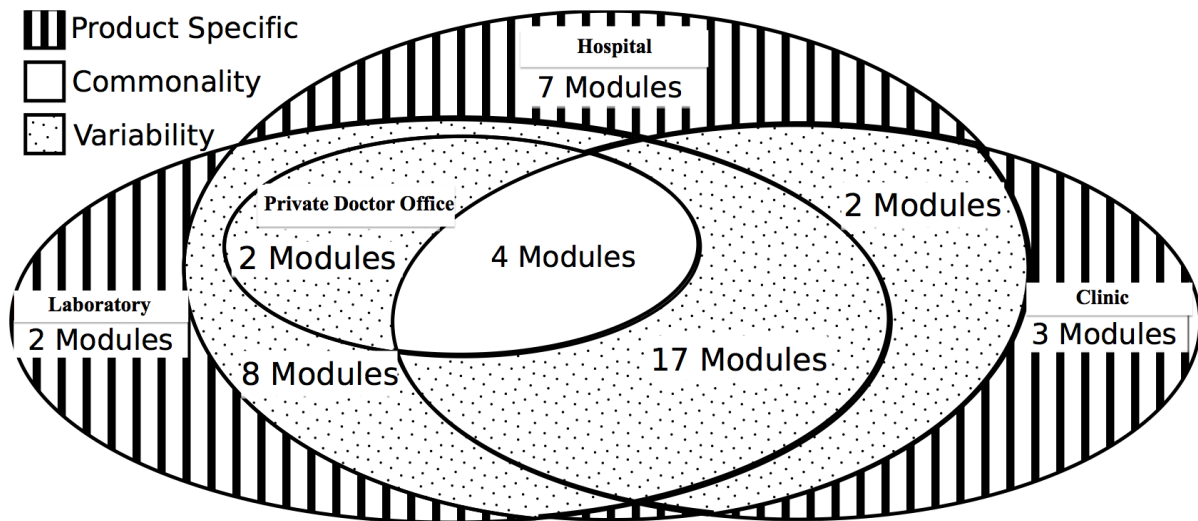


Figure 3.1: Modules (assets) per Areas Supported by MC.

we had about the collected data. To collect data from CIC we built a tool that accessed the CIC database and generated the necessary reports. To collect the data from LOC in PowerBuilder we used a software called Visual Expert³.

After collecting all the data, we started to group them according to an CIC filed. When registering a new request at CIC, the user must fill a field called *request type*. Based on this request type, the records from CIC were grouped according to the types of maintenance (Lientz and Swanson, 1980; Gupta et al., 2010). The records were grouped in three types of maintenance, according to Table 3.3.

It was possible to relate each request from the bug tracking system to either adaptive, corrective, or perfective maintenance since each request has a field for its type, and each type is related to a maintenance type. The preventive maintenance type was not used because none of the records corresponded to this type. We also show other records not related to maintenance types from CIC, since MC also use CIC to register management information. These other records had a null request type field or their request type field was related to commercial proposals, visiting requests or training requests. Thus, they were not used in the analysis. It was found a total of 70,652 requests in the CIC system.

Based on this classification and the LOC, it was possible to investigate each dependent variable and also perform the statistical analysis as discussed in the next section.

³<http://www.visual-expert.com/>

Table 3.3: Maintenance Types Groups at MC.

Maintenance Type	Request Type in CIC	Total of Records
Adaptive	Reports and System Adaptation Request	22,005
Corrective	System Error, Database Error, Operating System Error, Error Message of type "General Fail in the System", Error Message (Text in English) and System Locking / Freezing	14,980
Perfective	Comments, Suggestions and Slow System	2,366
Other	Doubts, Marketing - Shipping Material, Marketing-Presentation, Marketing-Proposal, Marketing-Negotiation, Training and Visit Request	31,301

3.3.2.2 Data Analysis and Discussion

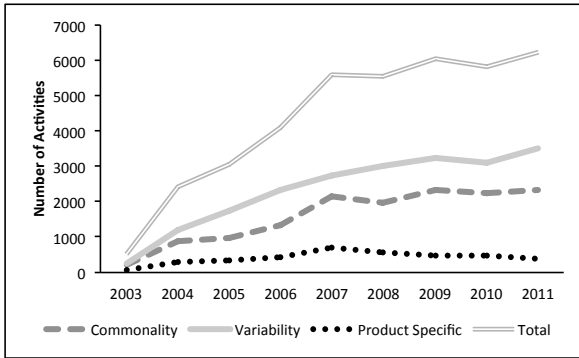
For analyzing the evolution at MC, in the first step, we collected data related to all assets and we did not distinguish common, variable and product-specific records. This step can be seen in each graph from Figure 3.2 as the *Total* line. As the objective was to evaluate the evolution in software product lines, the records were grouped into commonalities, variabilities and product-specific, facilitating the understanding of the evolution at MC.

The period in which the data were collected was not the same to all the laws evaluated. The continuing growth and conservation of familiarity laws were evaluated using data from the period between 1997 and 2011. The other laws (continuing change, increasing complexity, declining quality, and conservation of organizational stability) were evaluated using data from the period between 2003 and 2011.

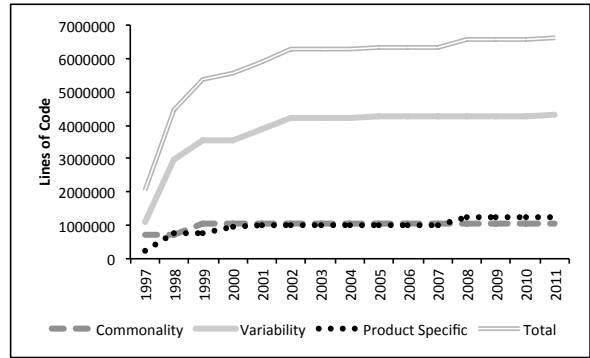
The descriptive statistics analysis and the discussion of this first empirical study results are shown next grouped by each research question.

RQ1.) Is there a relationship between the *Continuing Change* law and the evolution of common, variable, and product-specific assets?

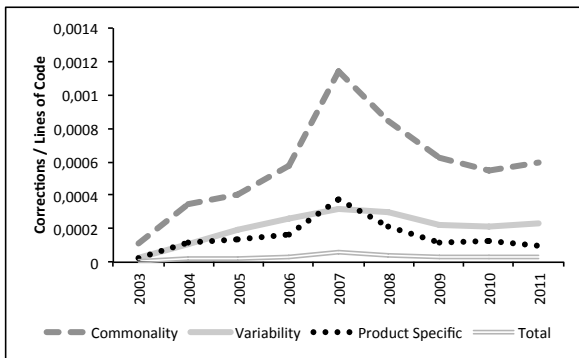
For this law, the number of activities (adaptive, corrective and perfective) registered in CIC from January 2003 up to December 2011 were used, corresponding to the *Number of Activities* dependent variable as shown on Figure 3.2(a). The plot shows a growth for the commonalities and variabilities, however, for the product-specific activities a small decrease can be noticed for the last years. The number of activities related to the variabilities are greater than the activities related to the commonalities. For the product-specific activities, as expected, there is a smaller number of activities, since it corresponds to the small group of assets from the MC SPL.



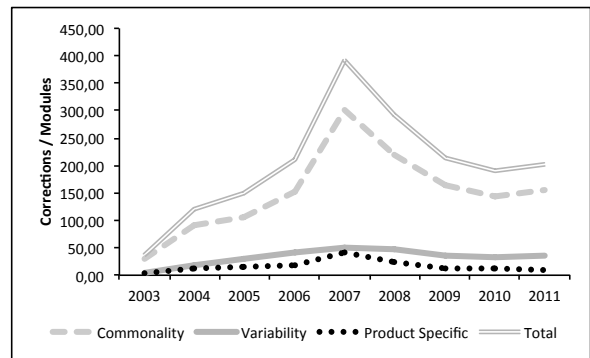
(a) Number of Activities



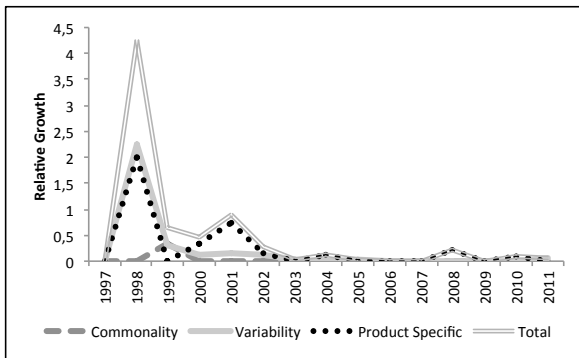
(b) Lines of Code per Year



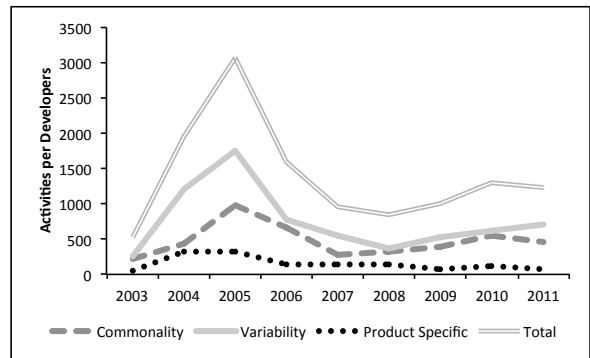
(c) Corrections / LOC per Year



(d) Number of Corrections per Module



(e) Relative Growth in Module Count



(f) Number of Activities per Developer

Figure 3.2: Plotted Graphs from CIC data and LOC (at MC)

Besides the small decrease for product-specific assets activities for the last years, we could identify a trend of growing in the number of activities for all assets (common, variable and product-specific) by applying the KPSS Test. Based on the confidence intervals analysis, Figure 3.3(a) (that presents, for each law, which asset – common, variable, and product-specific – corresponds more with a determined law) indicates that the different assets from the SPL have different amounts of activities. The number of activities related to the variabilities are bigger in the SPL because “*variability has to undergo continual and timely change, or a product family will risk losing the ability to effectively exploit the similarities of its members*” (Deelstra et al., 2004).

RQ2.) Is there a relationship between the *Continuing Growth* law and the evolution of common, variable, and product-specific assets?

Regarding this law, it was possible to identify similar behaviors for each of the SPL assets. Figure 3.2(b) shows the lines of code from 1997 up to 2011, corresponding to the *Lines Of Code* dependent variable. For common, variable and product-specific assets, we can observe a tendency to stabilization over the years. They grow at a high level in the first years, but they tend to stabilize over the next years.

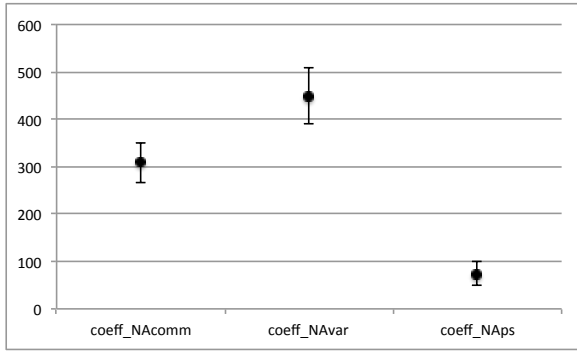
Due to the growth in the number of activities for common and variable assets according to the *Continuing Change* law, these activities had an impact on the LOCs. By using the KPSS Test, the commonalities and variabilities showed a trend of increasing. Despite the continuing change observed for the commonalities and variabilities in the SPL, MC does not worry about keeping the size of its common and variable assets stable, contributing with the increase of complexity.

For the product-specific assets, the KPSS Test showed a stationary behavior. Therefore, the *Continuing Growth* law to the product-specific assets is rejected. This happens because similar functions among the product-specific assets are moved to the core asset of the SPL (Mende et al., 2008).

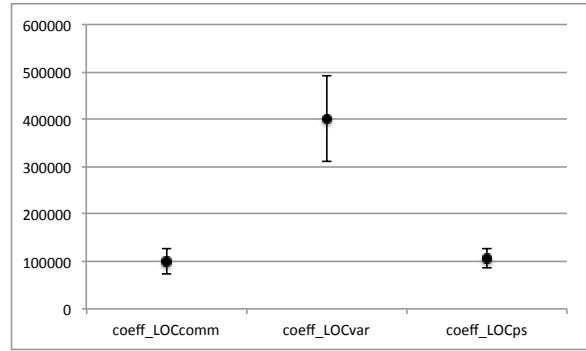
Moreover, through the confidence intervals analysis, Figure 3.3(b) shows that the variabilities from the SPL have more LOC than other assets. This could be a reason why variabilities have more activities (Continuing Change).

RQ3.) Is there a relationship between the *Increasing Complexity* law and the evolution of common, variable, and product-specific assets?

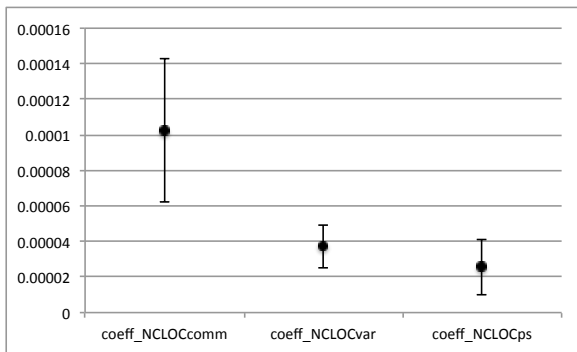
The total number of corrections per line of code, corresponding to the *Number of Corrections per LOC* dependent variable is shown in Figure 3.2(c). As it can be seen, the complexity for



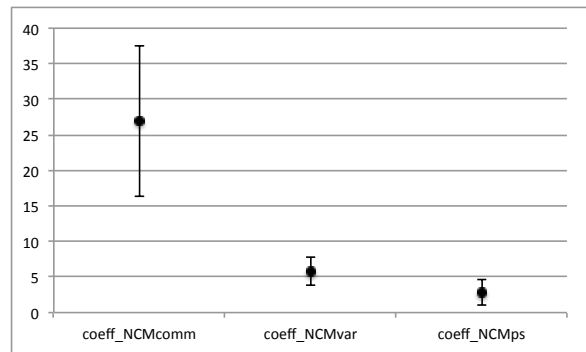
(a) NA



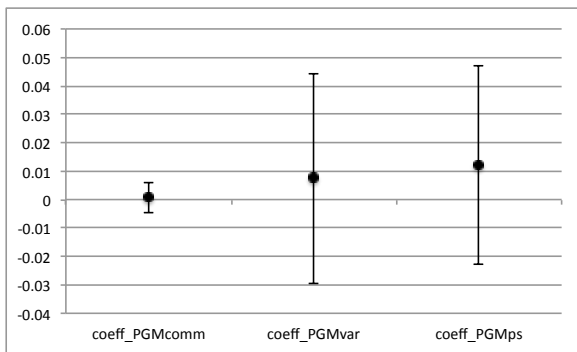
(b) LOC



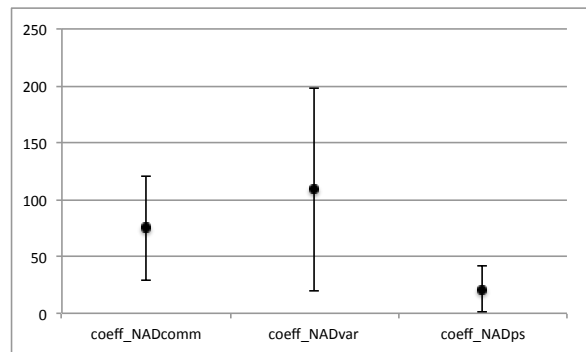
(c) NCLOC



(d) NCM



(e) RGM



(f) NAD

Figure 3.3: Confidence Intervals for the Regression Coefficients (at MC)

commonalities, variabilities and product-specific assets is increasing up to 2007. This increase was bigger for the commonalities because at that time MC had to evolve the SPL to support new government laws. However, variable and product-specific assets have also grown up to 2007, since modifications within common assets also had an impact on variable and product-specific assets (Svahnberg and Bosch, 1999; Ajila and Kaba, 2004; Bailetti *et al.*, 2004). After 2007, MC started to try to reduce the complexity and prevent the system from breaking down.

However, we could also identify a trend of growing in the complexity for the commonalities, variabilities and product-specific assets by applying the KPSS Test in the *Increasing Complexity* law. Hence, considering that the complexity is always growing, the *Increasing Complexity* law is supported for all the assets in the SPL at MC.

Confidence intervals analysis (see Figure 3.3(c)) indicates that the complexity inside the commonalities raises more than inside other assets. It happens because the commonalities have to support all the common assets from the products of the SPL and any change can affect the entire SPL (Svahnberg and Bosch, 1999; McGregor, 2003; Ajila and Kaba, 2004; Bailetti *et al.*, 2004).

RQ4.) Is there a relationship between the *Declining Quality* law and the evolution of common, variable, and product-specific assets?

The number of corrections per total of modules in the year, corresponding to the *Number of Corrections per Module* dependent variable, is shown in Figure 3.2(d). The number of corrections for the variabilities and for the specific assets follow almost the same pattern. However, for the common assets of the product line, we can notice a higher number of corrections per module in 2007, also caused by the adaptation of the system to the government laws. A small increase in the variabilities and in the specific assets also can be observed in the same year. From 2007, the number of corrections per module starts to decrease because of the feedback from users and corrections of problems related to the evolution to deal with the new government laws.

Besides the decrease after 2007, a trend of growing in the number of corrections per modules could be identified for the commonalities, variabilities and product-specific assets by using the KPSS Test. Hence, considering that the number of corrections per module is always growing, the *Declining Quality* law is supported for all the assets in the SPL at MC.

Based on the confidence intervals analysis, Figure 3.3(d), we could conclude that the number of corrections per module inside the commonalities is bigger than the other assets. As stated for commonalities the increasing complexity law, this happens because the commonalities have to support all the common assets from the products of the SPL and any change can affect the entire

product line (Svahnberg and Bosch, 1999; McGregor, 2003; Ajila and Kaba, 2004; Bailetti *et al.*, 2004).

RQ5.) Is there a relationship between the *Conservation of Familiarity* law and the evolution of common, variable, and product-specific assets?

The relative growth in module count, corresponding to the *Relative Growth in Module count* dependent variable, is shown in Figure 3.2(e). We can notice that in 1997, MC had already a well defined scope for the common assets since almost all of them were included in the SPL at that time. Over the years, most of the assets included in the SPL, for accommodating the necessary changes, were variable and product-specific assets.

Regarding this law, a similar behavior could be identified in the SPL assets. To support the Conservation of Familiarity law, we were looking for assets that have a stationary growth over the years. The relative growth in module count for common, variable and product-specific assets have a stationary behavior by applying the KPSS Test. Thus, the *Conservation of Familiarity* law was supported within this project.

Unfortunately, as shown in Figure 3.3(e), all the assets have intersections among their limits, thus, we could not conclude which asset had the biggest relative growth.

RQ6.) Is there a relationship between the *Conservation of Organizational Stability* law and the evolution of common, variable, and product-specific assets?

The total number of activities (adaptive, corrective, and perfective) per year divided by the total number of developers working on the activities, corresponding to dependent variable *Number of Activities per Developer*, is shown in Figure 3.2(f). The number of activities for the variabilities are bigger in the first years, however, it tends to decrease in the following years. This situation also happened within the common and product-specific assets of the product line.

To support the Conservation of Organizational Stability law, we were looking for assets that have a stationary growth over the years. Applying the KPSS Test in common, variable, and product-specific assets, the *Conservation of Organizational Stability* law was supported since they have a stationary trend, meaning that the conservation of organizational stability indeed exists.

Nevertheless, as shown in Figure 3.3(f), we could not conclude which assets had the biggest number of activities per developer, since the assets have intersections among their limits.

The results of the KPSS test for MC are shown in Appendix A.1.

Table 3.4: Laws and Results from the First Empirical Study (at MC).

Law	Commonalities	Variabilities	Product Specific
Continuing Change	Supported	Supported	Supported
Continuing Growth	Supported	Supported	Not Supported
Increasing Complexity	Supported	Supported	Supported
Declining Quality	Supported	Supported	Supported
Conservation of Familiarity	Supported	Supported	Supported
Conservation of Organizational Stability	Supported	Supported	Supported

From this first empirical study, commonalities, variabilities, and product-specific assets seems to behave differently regarding evolution. Five laws were completely supported (continuing change, increasing complexity, declining quality, conservation of familiarity, and conservation of organizational stability) in this empirical study. The other law (continuing growth) was partly supported, depending on the SPL asset in question, as shown in Table 3.4.

Next Section presents the motivation for performing the replicated empirical study.

3.3.3 Motivation for Conducting the Replication

The first empirical study was performed in the medical domain. In order to broaden the results of LL within SPL, the first empirical study was replicated in the financial domain (Oliveira *et al.*, 2015b). Some of the laws were completely supported in the first empirical study, such as *continuing change*, *increasing complexity*, *declining quality*, *conservation of familiarity*, and *conservation of organizational stability* laws. One law (*continuing growth*) was partially supported for common and variable assets and rejected for product specific ones. Since most of the laws were supported for the first empirical study, we started to understand how SPLs evolve and we would like to have more insights to propose improvements in the SPL evolution process.

This replication was conducted by the same researchers (internal replication), using the same research questions, dependent variables, metrics and hypotheses, however, as a replicated study, the domain was changed to draw further conclusions.

3.3.4 Changes to the Original Experiment

The first empirical study was evaluated using a larger data set, which was grouped by year. Within this first empirical study, it was possible to collect data since 1997 for LOCs and since 2003 for the bug tracking system. However, in the second study, the data set was smaller. Data from LOCs started to appear in January of 2009, and data from the bug tracking system started to

appear in November of 2010. Thus, in order to check some tendencies using statistical methods, the second study was grouped by months.

3.3.5 Second Empirical Study

The company (environment) in the replicated study was an IT company in the financial domain. Thus, herein to preserve the company's name, it will be called *Financial Company (FC)*. FC is using the SPL paradigm with success to develop their common, variable and product specific assets, and it has more than 3 years of historical data, which allowed the statistical data analysis. Thus, for FC we analyzed the data over the months.

FC builds products, using Delphi programming language⁴, for four main areas: *financial support*, *account support*, *budget support* and *fixed assets support*. During the investigation, FC allowed full access to its code and its bug tracking system.

Regarding the bug tracking system, FC uses one developed by FC itself, called *Client Relationship Center (CRC)*. CRC allows FC's users to register requests for adaptations, enhancements, corrections and also requests for the creation of new modules.

All the products from FC have some assets (called modules) in common (commonalities), some variable assets (variabilities) and also some specific assets, enabling the derivation of the specific products depending on the combination of the selected assets, characterizing the FC SPL.

Figure 3.4 shows the division of modules among the main areas in which FC develops products. One (1) module composes the commonalities of the FC SPL, seven (7) modules compose the variabilities of the FC SPL, and the others eighteen (18) modules compose the product specific assets, totalizing twenty-six (26) modules in the FC SPL. These modules were classified into common, variable, and product-specific according to their usage in the FC supported areas. A senior developer helped us with the classification, since these modules did not have clear boundaries among them.

Based on data of those modules, most of the laws could be evaluated with the records from CRC, however, other ones required the LOC metric. From CRC, it was collected data since 2010 concerning to the three types of maintenance (adaptive, corrective, and perfective). Regarding LOC, it was possible to collect data since 2009.

⁴<http://www.embarcadero.com/products/delphi>

3.3.5.1 Execution

The object of the second empirical study was the FC SPL. It was defined an approach composed of two steps to collect the necessary data (from source code and the bug tracking system). In the first step, it was collected data from CRC and LOC. These data correspond to all types of requests that the users can make within CRC and the total LOCs, respectively. In a second step, FC clarified some doubts, through interviews, about the collected data. To collect data from CRC, reports were exported by the tool. The developers extracted the information from LOC in Delphi and sent to us.

After collecting all the data, we started to group them according to an CRC field. When registering a new request, the user must fill a field called *request type*. Based on this request type, the records from CRC were grouped according to the types of maintenance (adaptive, corrective, and perfective), as shown in Table 3.5.

It was possible to relate each request from the bug tracking system to either adaptive, corrective, or perfective maintenance since each request has a field for its type, and each type is

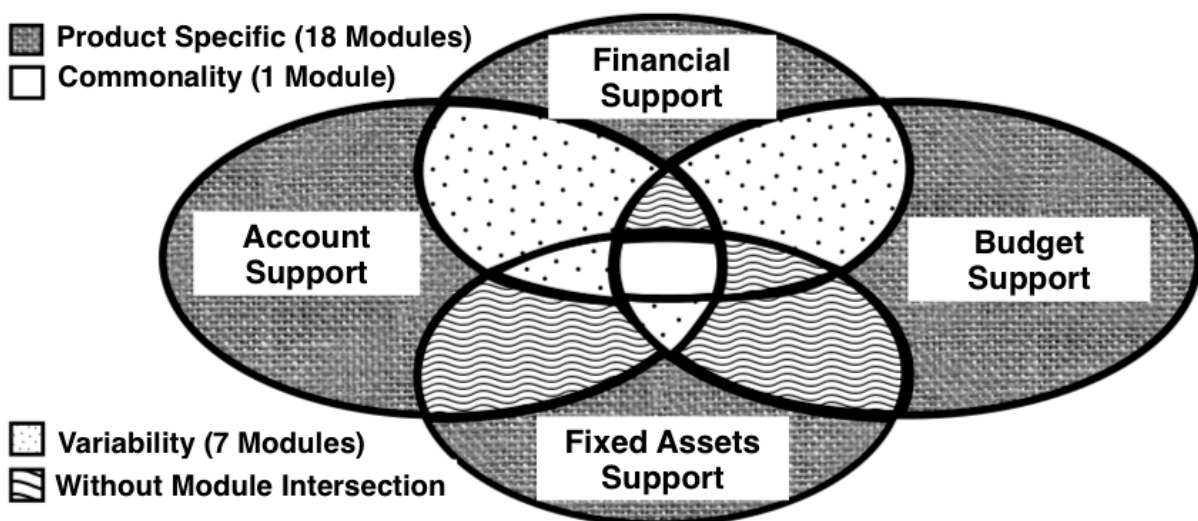


Figure 3.4: Modules (assets) per Area at FC.

Table 3.5: Maintenance Types Groups at FC.

Maintenance Type	Total of Records
Adaptive	25
Corrective	724
Perfective	41
TOTAL	790

related to a maintenance type. Thus, it was investigated each dependent variable and performed statistical analysis based on this classification and the LOC, as discussed in the next section.

3.3.5.2 Data Analysis and Discussion

For analyzing the evolution at FC, in the first step, it was collected data related to all assets and it was not distinguished common, variable, and product-specific records to check the general evolution of FC assets. This step can be seen in each graph from Figure 3.5 as the *Total* line. As the objective was to evaluate the evolution within SPL, the records were grouped into commonalities, variabilities, and product-specific facilitating the understanding of the evolution at FC.

The period in which the data were collected was not the same to all the laws evaluated. The continuing growth and conservation of familiarity laws were evaluated using data from the period between January of 2009 and April of 2012. The other laws (continuing change, increasing complexity, declining quality, and conservation of organizational stability) were evaluated using data from the period between November of 2010 and June of 2012.

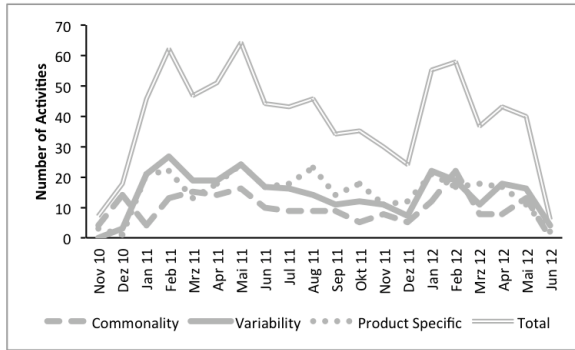
The descriptive statistics analysis and the discussion of the second empirical study results are shown next, grouped by each research question.

RQ1.) Is there a relationship between the *Continuing Change* law and the evolution of common, variable, and product-specific assets?

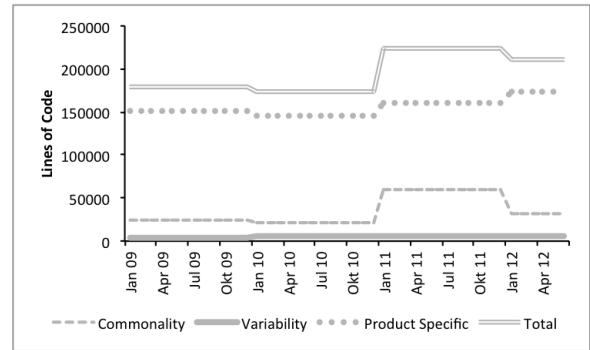
For this law, we used the number of activities (adaptive, corrective, and perfective) registered in CRC from November 2010 up to June 2012, corresponding to the *Number of Activities* dependent variable as shown on Figure 3.5(a). The plot shows that all assets (common, variable, and product specific) grow up in the first months, however, after that, they present a ripple effect and they do not grow at all over the next months. By applying the KPSS statistical test, we could identify that commonalities, variabilities, and product specific assets have a stationary behavior over the months. Thus, we could not support the *continuing change* laws for all the assets at FC.

Based on the confidence intervals analysis, Figure 3.6(a) (that presents, for each law, which asset – common, variable, and product-specific – corresponds more with a determined law), we could not identify which asset had more activities since they have intersections among them.

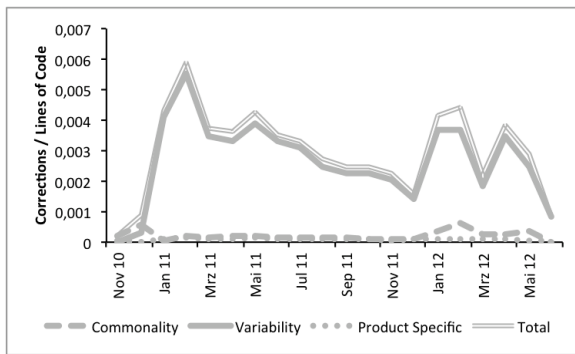
RQ2.) Is there a relationship between the *Continuing Growth* law and the evolution of common, variable, and product-specific assets?



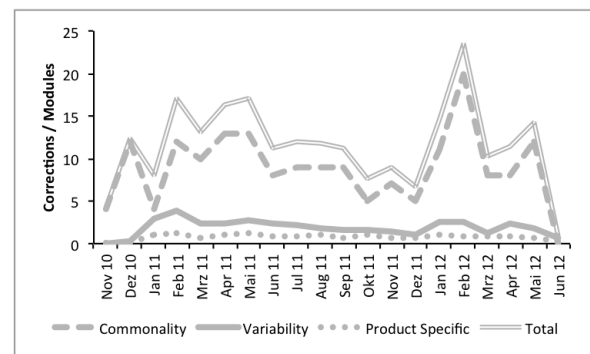
(a) Number of Activities



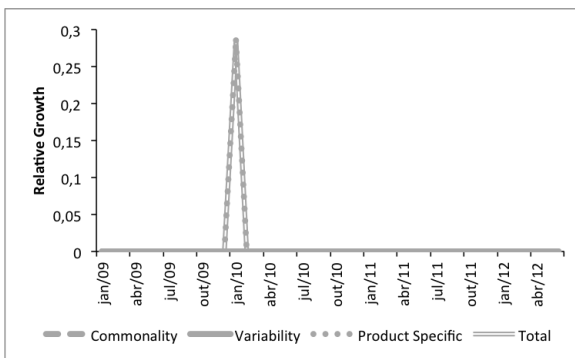
(b) Lines of Code



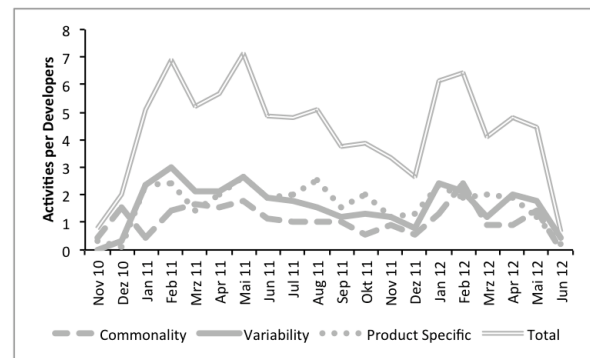
(c) Corrections / LOC



(d) Number of Corrections per Module

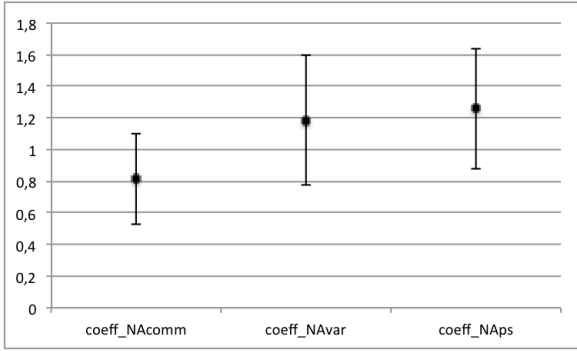


(e) Relative Growth in Module Count

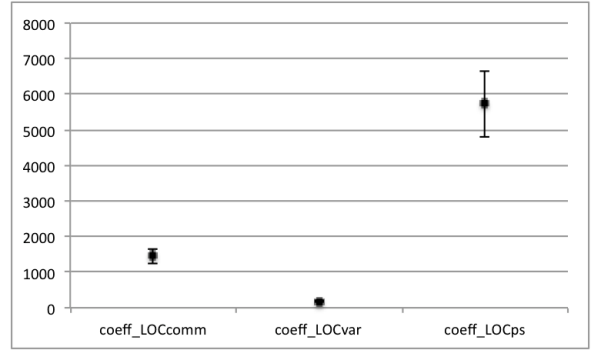


(f) Number of Activities per Developer

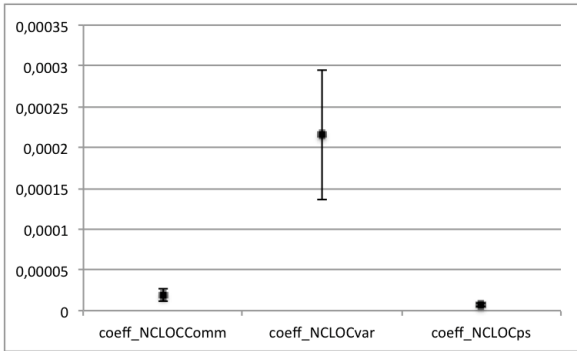
Figure 3.5: Plotted Graphs from Bug Tracking System data and LOC (at FC)



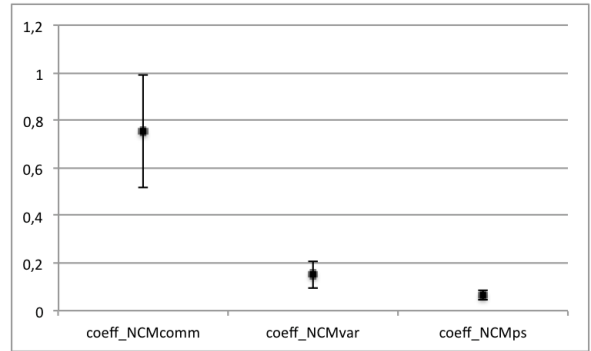
(a) NA



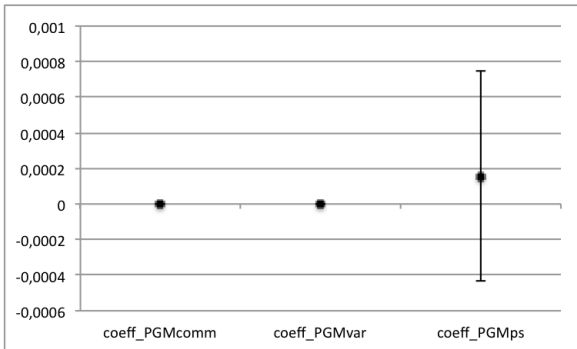
(b) LOC



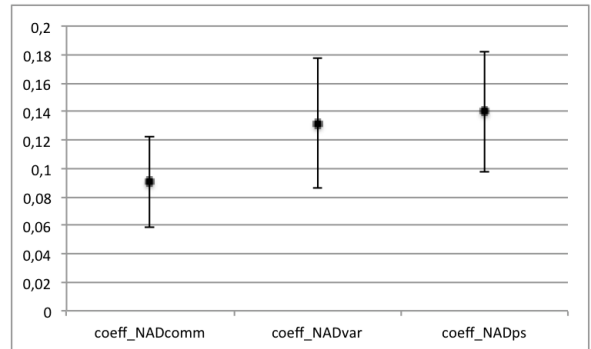
(c) NCLOC



(d) NCM



(e) RGM



(f) NAD

Figure 3.6: Confidence Intervals for the Regression Coefficients (at FC)

Regarding this law, it was possible to identify similar behaviors for each of the SPL assets. Figure 3.5(b) shows the lines of code from January of 2009 up to April 2012, corresponding to the *Lines Of Code* dependent variable. For common, variable, and product-specific assets, we can observe a small tendency of growing over the months. Besides, there is no such growing in the activities according to the *continuing change* law, overall, the total LOC of each asset had a growth trend.

By using the KPSS Test, the commonalities, variabilities, and product specific assets showed a trend of increasing, thus, we could support the *continuing growth* law for all assets.

Moreover, through the confidence intervals analysis, Figure 3.6(b) shows that the product specific assets had more LOC than other assets. One of the reasons is that product specific assets correspond to the majority of assets at FC SPL.

RQ3.) Is there a relationship between the *Increasing Complexity* law and the evolution of common, variable, and product-specific assets?

The total number of corrections per line of code corresponds to the *Number of Corrections per LOC* dependent variable, as shown in Figure 3.5(c). As it can be seen, the complexity for variabilities is higher compared to other assets. However, we could not identify a trend of growing in the complexity for the commonalities, variabilities and product-specific assets by applying the KPSS Test in the *Increasing Complexity* law. Hence, considering that the complexity is not growing over the months, the *Increasing Complexity* law is not supported for all the assets in the FC SPL.

Confidence intervals analysis (see Figure 3.6(c)) indicates that the complexity inside the variabilities raises more than inside other assets. It happens because “*variability has to undergo continual and timely change, or a product family will risk losing the ability to effectively exploit the similarities of its members*” (Deelstra et al., 2004). These continual and timely changes raise the complexity of variabilities at FC, however, the company deals with this complexity avoiding its growth.

RQ4.) Is there a relationship between the *Declining Quality* law and the evolution of common, variable, and product-specific assets?

The number of corrections per total of modules in the year, corresponding to the *Number of Corrections per Module* dependent variable, is shown in Figure 3.5(d). The number of corrections for the variabilities and for the specific assets follow almost the same pattern. However, common assets of the SPL had a higher number of corrections per module. Besides this growth for

common assets, the commonalities, variabilities, and product-specific assets had a stationary behavior over the months by using the KPSS Test. Hence, considering that the number of corrections per module is not growing, the *Declining Quality* law is not supported for all the assets in the FC SPL.

Based on the confidence intervals analysis, Figure 3.6(d), we could conclude that the number of corrections per module inside the commonalities is bigger than the other assets. As stated, this happens because the commonalities have to support all the common assets from the products of the SPL and any change can affect the entire product line (Svahnberg and Bosch, 1999; McGregor, 2003; Ajila and Kaba, 2004; Bailetti *et al.*, 2004).

RQ5.) Is there a relationship between the *Conservation of Familiarity* law and the evolution of common, variable, and product-specific assets?

The relative growth in module count, corresponding to the *Relative Growth in Module count* dependent variable, is shown in Figure 3.5(e). We can notice a significant growth in the first months of 2010 for product specific assets. For common and variable assets we could notice an stationary behavior over the months.

To support the Conservation of Familiarity law, we were looking for assets that have a stationary growth over the years. The relative growth in module count for common, variable could not be evaluated because of the stabilization over the months. Product-specific assets have a stationary behavior by applying the KPSS Test. Thus, the *Conservation of Familiarity* law was supported only within the product specific assets.

Unfortunately, as shown in Figure 3.6(e), all the assets have intersections among their limits, thus, we could not conclude which asset had the biggest relative growth.

RQ6.) Is there a relationship between the *Conservation of Organizational Stability* law and the evolution of common, variable, and product-specific assets?

The total number of activities (adaptive, corrective and perfective) per year divided by the total number of developers working on the activities, corresponding to dependent variable *Number of Activities per Developer*, is shown in Figure 3.5(f). The number of activities for the variabilities are bigger in the first months, however, it tends to decrease in the following months. This situation also happened within the common and product-specific assets of the product line.

To support the Conservation of Organizational Stability law, we were looking for assets that have a stationary growth over the months. Applying the KPSS Test in common, variable and product-specific assets, the *Conservation of Organizational Stability* law was supported since

Table 3.6: Laws and Results from the Second Empirical Study (at FC).

Law	Commonalities	Variabilities	Product Specific
Continuing Change	Not Supported	Not Supported	Not Supported
Continuing Growth	Supported	Supported	Supported
Increasing Complexity	Not Supported	Not Supported	Not Supported
Declining Quality	Not Supported	Not Supported	Not Supported
Conservation of Familiarity	-	-	Supported
Conservation of Organizational Stability	Supported	Supported	Supported

they have a stationary trend, meaning that the conservation of organizational stability indeed exists.

Nevertheless, as shown in Figure 3.6(f), we could not conclude which assets had the biggest number of activities per developer, since the assets have intersections among their limits.

The results of the KPSS test for FC are shown in Appendix A.2.

From this second empirical study, commonalities, variabilities, and product-specific assets seems to behave differently regarding evolution. Two laws were completely supported (continuing growth and conservation of organizational stability) in this empirical study. Three laws were not supported (continuing change, increasing complexity, and declining quality). Finally, one law (conservation of familiarity) was supported only for product-specific assets, as shown in Table 3.6.

Next, it is presented a comparison of the results from both empirical studies.

3.3.6 Comparison and Discussion of Results

After applying the KPSS statistical test in both empirical studies, it was found that the results are one-third consistent. Next, it is discussed the consistent and different results, according to Table 3.7.

Table 3.7: Consistent/Different Results from the Empirical Studies.

Dependent Variable	Law	Commonalities	Variabilities	Product Specific
NA	Continuing Change	Different	Different	Different
LOC	Continuing Growth	Consistent	Consistent	Different
NCLOC	Increasing Complexity	Different	Different	Different
NCM	Declining Quality	Different	Different	Different
RGM	Conservation of Familiarity	Different	Different	Consistent
NAD	Conservation of Organizational Stability	Consistent	Consistent	Consistent

3.3.6.1 Consistent Results

Comparing both empirical studies, one law (*conservation of organizational stability*) was completely supported for all assets (commonalities, variabilities, and product specific), as shown in Table 3.7.

- **The *Conservation of Organizational Stability (NAD)* law.** For this law, our findings shown that common, variable, and product specific assets have a stable behavior over the years for both empirical studies. Thus, the number of activities per developer over the time for both companies are constant, supporting this law.

Moreover, since the number of activities in all assets per developers keeps constant, these companies avoid a frequent problem of overloading the developers with extra tasks, smoothing the SPL evolution.

3.3.6.2 Partially Consistent Results

We also had partially consistent results for some assets concerning two laws (*continuing growth* and *conservation of familiarity*).

- **The *continuing growth (LOC)* law.** We could support the *Continuing Growth (LOC)* law in both empirical studies for common and variable assets, meaning that these assets keep growing in LOC over the years. Moreover, according to the first study at MC, variable assets had more LOC than other assets to support all the product configurations (Figure 3.3(b)).

The increase of LOC for common and variable is an evidence that a control under the growth of these assets should exists to avoid a future increase within the SPL complexity.

- **The *Conservation of Familiarity (RGM)* law.** Product specific assets for this law also had a consistent result. Both results showed a stationary trend in the relative growth of module count over the time. Thus, this law was supported for product specific assets within both empirical studies.

3.3.6.3 Partially Different Results

We also had partially different results for some assets concerning the *continuing growth* and *conservation of familiarity* laws.

- **The *continuing growth (LOC)* law.** Product specific assets had a difference in the results for both empirical studies according to this law. With the MC empirical study, this law was not supported. However, it was supported for product specific assets at the second empirical study in the FC. Thus, the number of LOC's for product specific assets within FC increased over

the time being also responsible for the higher number of LOC's within this SPL compared to other assets (Figure 3.6(b)). This can also bring some increase in their complexity in a near future. Thus, in order to avoid such increasing, a standard behavior within an SPL should be a refactoring of product specific assets into common or variable ones.

- **The Conservation of Familiarity (RGM) law.** According to this law, the system growth should be constant. This could be confirmed for common and variable assets of the first empirical study. Hence, there is no relative growth in module count over the time. However, common and variable assets of the second study at FC could not be evaluated through the KPSS statistical test since the data were constant over the time. Thus, we need more studies to support this law for SPL common and variable assets.

3.3.6.4 Differences in Results

Three of the laws (*continuing change*, *increasing complexity*, and *declining quality*) had differences within all the assets (common, variable, and product specific) according to the results of both empirical studies, as shown in Table 3.7. Next, it is discussed the differences.

- **The Continuing Change (NA) law.** This law was completely supported for all assets in the first empirical study at MC. Hence, the number of activities for MC is increasing over the time. It is interesting to check that the number of activities for the medical domain are increasing, thus the need to evolve the SPL according to the user needs or due a changing environment indeed happens. On the other side, neither asset supported this law within the second empirical study at FC. In the financial domain, the need to evolve the SPL according to the user needs does not happen because of the maturity in the domain, where common, variable, and product-specific assets were strictly developed according to user needs.

For the first study, we could identify that variable assets have the greater number of activities (Figure 3.3(a)). The number of activities for variable and product specific assets in the second study seems to be apparently higher than activities for common assets, however, we could not verify which asset had more activities, since they had intersections among their limits, as shown in Figure 3.6(a). Nevertheless, we should consider the number of activities for variable assets, since it was the biggest number within the first study and it was also high for the second one. This may happen to keep updated the variabilities within the SPL, allowing the configuration, and customization of several products.

- **The Increasing Complexity (NCLOC) law.** This law was completely supported for all assets in the first empirical study at MC. Hence, the number of corrections per LOC in MC is increasing over the time. On the other side, neither asset supported this law within the second

empirical study at FC.

Moreover, the complexity was higher for common assets within the first empirical study (Figure 3.3(c)), and it was higher for variable assets within the second empirical study (Figure 3.6(c)). The complexity for these assets may be higher because of the following SPL behaviors: common assets should support all the products from the SPL (McGregor, 2003) and; “*variability has to undergo continual and timely change, or a product family will risk losing the ability to effectively exploit the similarities of its members*” (Deelstra *et al.*, 2004).

- **The Declining Quality (NCM)**. This law was completely supported for all assets in the first empirical study at MC. Thus, the number of corrections per module count within MC is increasing over the time. Nonetheless, neither asset supported this law within the second empirical study at FC.

For both empirical studies, the evidences shown that common assets had more declining quality than others (Figures 3.3(d) and 3.6(d)). Since commonalities have to support all the common assets from the SPL products, their quality also should rise to improve the whole SPL quality. Next, it is presented the threats to validity of the studies.

3.4 Threats to Validity

There are some threats to the validity of the study. They are described and detailed as follows.

External Validity threats concern the generalization of our findings. It was analyzed SPLs from two specific domains: the medical and the financial ones. There are requests for changing that are specific to these domains such as the need of changing because of a new government law (medical domain). Hence, it is not possible to generalize the results to other domains than the ones investigated here. However, these are the first results, which can be considered within the medical and financial domains.

These empirical studies involved two SPL companies. Thus, it needs a broader evaluation in order to try to generalize the results. However, this was the first step where LL were evaluated within two industrial SPL from different domains in which a long historical data was made available for the researchers.

Internal Validity threats concern factors that can influence our observations. The period in which the data were collected was not the same. Some laws were evaluated using the period between 1997-2011 (MC) / 2009-2012 (FC). Others were evaluated using the period between 2003-2011 (MC) / 2010-2012 (FC). For MC we analyzed data by year and for FC we analyzed data by months. Also, the requests from the bug tracking system were used in the same way no

matter of their quality, duplication, request implementation, or rejected requests. Moreover, the size in modules of the systems were not the same and they did not have clear boundaries among them. Nevertheless, we asked a senior developer to help us with the modules classification. Moreover, the available data are meaningful because we could deal with industrial SPL projects with long periods of historical data, where statistical methods could be successfully applied.

Construct Validity threats concern the relationship between theory and observation. The metrics used in this study may not be the best ones to evaluate some of the laws, considering that there is no baseline for those metrics applied to SPL. However, metrics used to evaluate LL in previous studies were the basis for this work. Some metrics were based on LOC. Even though LOC can be considered a simplistic method, LOC and Cyclomatic Complexity are found to be strongly correlated (Kan, 2002), thus we decided to use LOC since MC and FC had this information previously available.

Conclusion Validity threats concern with issues that affect the ability to draw the correct conclusion about the outcome of the study. Indeed, not all of the laws were supported to all assets (common, variable, and product-specific) concerning the evaluated laws. However, most of the laws were supported according to the asset type within both empirical studies (twenty-four supported assets from a total of third-six assets).

3.5 Key Findings and Contributions for SPL Community

In this section, it is presented the key findings and also discussed what is the impact of each finding for industrial SPL practitioners, according to each law.

- a. *Continuing Change Law.* Variable assets are responsible for the greater number of activities performed in the MC industrial SPL project and also it is one of the assets with more activities for the FC SPL industrial project. Practitioners should be aware of making modifications within those assets, since there are several constraints among them. According to the MC SPL, we could realize that the number of product-specific activities decreases starting 2007 while the number of activities on common and variable assets increases. It could be that there are so many activities on the variable and common assets (compared to the product-specific assets) because their scope has not been chosen well (or has changed significantly in 2007), implying that more and more specific assets have to be integrated into commonalities and variabilities. This would be a typical SPL behavior. Also, another reason for increasing the number of activities on variable assets is that SPL needs more attention within the variabilities for the sake of reuse.

-
- b. *Continuing Growth Law*. The increase of LOC for common and variable is an evidence that a control under the growth of these assets should exist to avoid a future increase within the SPL complexity. Also, we identified that variable and product-specific assets had also the biggest growth. Practitioners should search among the variable and product specific assets, those that share behavior and can be transformed into common assets. Transforming variable and product specific assets into common assets will reduce their total growth and it also will reduce their complexity.
 - c. *Increasing Complexity Law*. Complexity within common assets is bigger than for other assets within the MC SPL. According to the FC SPL, variable assets had the biggest complexity. Practitioners should be aware of complexity in common and variable assets since they have to support all the products and variations from the SPL (McGregor, 2003; Deelstra et al., 2004). This makes any kind of change in common and variable assets to be considered as critical, since they may affect the whole SPL.
 - d. *Declining Quality Law*. The number of corrections per modules were higher for common assets in both studies. In fact, for these empirical studies we also have to consider the number of maintainers at both companies, which were a small number. Moreover, it is very important to control the quality of the SPL, mainly for commonalities. Since they have to support all the common assets from the SPL, their quality also should rise to improve the whole SPL quality.
 - e. *Conservation of Familiarity Law*. All assets from MC (common, variable and product-specific) supported the conservation of familiarity law. Within the FC SPL, we could not evaluate this law for common and variable assets because of the data stabilization over the time. Hence, more studies are needed. However, product specific assets at FC also supported this law. In order to avoid the loss of familiarity with the SPL, practitioners should maintain the relative growth for all assets constant, avoiding the exponential growth.
 - f. *Conservation of Organizational Stability Law*. All assets support this law by keeping the same amount of work among the developers for both empirical studies. Practitioners should try to achieve a better management of the developers' work, without overloading them with unnecessary work.

Based on the results of these empirical studies, the following initial items were proposed to improve the evolution within SPLs:

- . *Creation of guidelines for evolving each SPL artifact.* The lack of systematization over the SPL evolution process may led to inconsistent results for common, variable, and product-specific assets among the companies. Thus, guidelines supporting evolution steps for SPL artifact should exist to systematize the evolution of common, variable and product-specific assets. These guidelines should consider why, when, where and how the SPL assets evolve.
- . *For each evolution task, keep constant or improve the quality of the SPL.* Although the results from the KPSS statistical test did not confirm the loss of quality for the FC company, the regression analysis shown that common assets within the FC are losing quality. Thus, measurements should be applied over the evolution process within the all the SPL assets. These measurements can be extended also to all phases of the SPL development (including requirements, architecture, code, and so on).
- . *For each evolution task, try to decrease the complexity of the SPL.* Although the results from the KPSS statistical test did not confirm the increase of complexity for the FC company, the regression analysis shown that the complexity of variable assets within the FC is increasing. Thus, after evolving the SPL code, measurements may be applied to check if the new change in the code increases or not the complexity of the SPL;
- . *Improve the feedback.* Our results shown that both companies care about the amount of work performed by developers. This is a good practice within the SPL development. Thus, the management team should be aware of the clients' needs in order to keep adding/removing functionalities within the SPL paying attention to not increase LOCs of the assets.

The above list summarizes some improvements that can be followed according to the findings of these empirical studies at MC and FC. Next Section presents the Chapter summary.

3.6 Chapter Summary

Lehman's Laws of Software Evolution were published in the seventies, updated years ago, and are still perceived in nowadays software evolution context. The investigation performed here with two industrial SPL projects shown that commonalities, variabilities, and product-specific assets seems to behave differently regarding evolution.

Few results were consistent between both empirical studies. The benefits are twofold: the consistent results confirm once again the LL regarding SPL; and the different results indicate

that more investigation is needed to better understand the reason why LL may not hold for SPL. However, we believe that the LL are also applicable in the SPL context, since most of the laws could be supported for most of the SPL assets of the two SPL industrial projects (twenty-four supported assets from a total of third-six assets).

In summary, the consistent results shown that SPL support the conservation of organizational stability law, however, there is a need to cope with the continuing growth, which may affect the whole SPL complexity and quality. According to the first study, all assets are changing over the time. However, there is an increasing of complexity and a decrease of quality over the years. Therefore, dealing with the complexity and quality in evolving an SPL needs special attention. Additionally, guidelines may help with the declining quality and increase complexity during the SPL evolution, since they can make the evolution more effective. These guidelines may help during the whole SPL evolution starting from the SPL requirements up to the SPL tests, considering all LL of software evolution. Next Chapter presents a systematic mapping study on SPL evolution, which identified approaches from 1996 up to the end of 2014.

4

Software Product Lines Evolution: A Systematic Mapping Study

Software Product Lines (SPL) evolution is a highly active research area, which receives significant attention of the software engineering community. However, so far, there is no systematic overview including this whole research area. Thus, this Chapter presents an overview from the state of research on SPL evolution. It was performed a systematic mapping study on SPL evolution which selected relevant papers from 1996 up to the end of 2014.

The remainder of this Chapter is organized as follows: Section 4.1 introduces the concept of systematic mapping study and presents the motivation to perform this study. Section 4.2 presents the general background and relates our study to others (Section 4.2.1). Section 4.3 shows the research method applied in this systematic mapping study. Section 4.4 discusses the results according to each defined research question and presents the threats to validity. Section 4.5 presents the Chapter summary and the consolidation of the findings from both empirical studies, which evaluated the applicability of LL within SPL, and the systematic mapping study.

4.1 Introduction

A systematic mapping study provides well defined procedures to identify work related to a research question. Moreover, it also reveals gaps in current research in order to foster further investigation (Budgen *et al.*, 2008).

Thus, in order to understand and improve the Software Product Lines (SPL) evolution processes, a mapping study was performed (Oliveira *et al.*, 2015d). The focus of this study is to reveal approaches that deal with SPL evolution and reveal gaps for future research.

4.1.1 Motivation

So far, there is still a lack of a global landscaping covering the whole SPL life cycle and its evolution. A systematic mapping study can help to reveal how approaches are dealing with SPL evolution and show some gaps that need to be improved.

To the best of our knowledge, there are only two systematic mapping study about SPL evolution: [Laguna and Crespo \(2013\)](#) and [Assunção and Vergilio \(2014\)](#). However, these mapping studies focuses only in re-engineering into SPLs and SPL refactoring.

Thus, the proposed systematic mapping study is going to be the first one to identify:

- approaches that deal with evolution in SPL
- why, where, when, what, and how SPL evolve

Moreover, there are some benefits of this systematic mapping study, as follows:

- *This study can benefit researchers, practitioners.* Since it presents a whole overview of the SPL evolution area, from 1996 up to the end of 2014, researchers and practitioners may choose the best SPL evolution approach according to their needs.
- *Point out research gaps.* This study pointed out some research gaps that so far were not researched or need more investigation.
- *Since this study is based on a taxonomy for software change ([Buckley et al., 2005](#)), it can be possible to suggest a taxonomy for SPL evolution.* This taxonomy should take into account where most of the studies fitted according to the data extraction form and try to identify what is specific to SPL.

4.2 Background

There are some taxonomies ([Buckley et al., 2005](#); [Fenske et al., 2014](#)) and classifications ([Bosch and Ran, 2000](#)) for evolution, which served as a basis to this work.

[Fenske et al. \(2014\)](#) built a taxonomy for SPL re-engineering. They gave different names to distinct activities and shown their relationships. However, their effort was only concentrated in the categorization of existing work. Moreover, since their proposed taxonomy only deals with SPL re-engineering, some relevant SPL evolution approaches may not fit into the taxonomy. Thus, we decided to use the more general taxonomy proposed by [Buckley et al. \(2005\)](#).

[Buckley et al. \(2005\)](#) defined a taxonomy for software changes within single systems. This taxonomy identifies *when, where, what, and how* a single system evolves, according to: the

Temporal Properties Evaluated (When), the *Object of Change Evaluated (Where)*, the *System Properties (What)*, and the *Change Support (How)*. However, this taxonomy has not been applied to SPL so far. Thus, within this systematic mapping study, we use this taxonomy to identify *When*, *Where*, *What*, and *How* SPL evolve over the time.

Bosch and Ran (2000) defined some SPL categories of SPL evolution, as follows: *New product family*; *Introduction of New Product*; *Adding New Features*; *Extend Standard Support*; *New Version of Infrastructure*; and *Improvement of Quality Attribute*. Based on these categories of SPL evolution, we could identify within the selected approaches *Why* an SPL needs to evolve.

4.2.1 Related Work

To the best of our knowledge this is the first systematic mapping study covering the whole SPL evolution area. However, there are other mapping studies on SPL evolution that cover specific areas (Laguna and Crespo, 2013; Assunção and Vergilio, 2014; Santos *et al.*, 2015a). Next it is present an overview of these studies.

SPL evolution is an highly active research area which have already some systematic mapping studies. Laguna and Crespo (2013) performed a systematic mapping study which focused on approaches that deal with re-engineering of legacy system to SPL and refactoring of SPL. Assunção and Vergilio (2014) also performed a systematic mapping study, however their focus was only within migrating a single system to SPL through the identification of features. Since the focus from these work were limited (re-engineering and refactoring), some SPL evolution studies remain unrevealed. Moreover, within their search string, the authors did not consider the term *evolution*. Santos *et al.* (2015a) mapped the existing approaches dealing with inconsistency management within SPL. Such approaches cope with the identification and management of inconsistencies during the evolution of SPL artifacts. They classified and performed a characterization of these approaches through a mapping study, and organized it in three main categories: (i) Model against Source code, (ii) Model against Model, and (iii) Model against Specifications. However, their mapping study focused only in approaches that deal with inconsistencies within the evolution of SPL artifacts. Thus, our systematic mapping study aims to complement these SPL mapping studies by addressing all approaches concerning SPL evolution.

4.3 Research Method

We followed the guidelines provided by [Budgen *et al.* \(2008\)](#); [Petersen *et al.* \(2008\)](#); [Kitchenham and Charters \(2007\)](#). A systematic mapping study is a way of categorizing and summarizing the existing information about a research question in an unbiased manner. This one was performed in three stages: Planning, Conducting, and Reporting (this document). The Planning and the Conducting stages are presented next.

4.3.1 Planning Stage

This stage is composed of the following activities: definition of the research question (Section 4.3.1.1); definition of the search strategy (Section 4.3.1.2); selection of primary studies (Section 4.3.1.3); definition of the quality assessment form (Section 4.3.1.4); and definition of the data extraction strategy (Section 4.3.1.5).

4.3.1.1 Research Question

The goal of this study is to reveal approaches that deal with evolution in SPL from the point of view of the following research question: ***RQ1) Which are the existing approaches for dealing with evolution in SPL?*** This research question allowed us to categorize and summarize the current knowledge concerning SPL evolution approaches, and also, to identify gaps for further investigation. Since the main research question is too broad, we decomposed it into more detailed sub-questions. Table 4.1 shows nine sub-research questions and their motivations.

4.3.1.2 Search Strategy

We used four digital libraries to perform the automated search for primary studies, as follows: IEEE *Xplore*; ACM Digital Library; Springer Link; and Science Direct. Moreover, we also manually searched in the main conference proceedings and journals that are relevant for SPL and evolution. Table 4.2 shows the manual searched conferences and Table 4.3 shows the manual searched journals. Although some conferences and journals selected for the manual search are indexed by some of the selected digital libraries, we also performed the manual search to ensure that all relevant papers have been included in the systematic mapping study, since there may be a bias in the automated search ([Santos *et al.*, 2015b](#)).

To perform the automatic search within the digital libraries, we built a search string consisting of two parts focusing on covering the concepts that represent the evolution in SPL. The first

Table 4.1: Sub-Research Questions and Motivation.

Sub-Research Questions	Motivation
<i>RQ1.1) Why</i> SPL approaches need to deal with evolution?	Identify the reason <i>Why</i> SPL evolve over the time according to Bosh and Ran SPL evolution categories (Bosch and Ran, 2000).
<i>RQ1.2) When</i> SPL approaches perform the evolution?	Identify <i>When</i> SPL need to evolve according to the taxonomy defined by Buckley <i>et al.</i> (2005). This includes the time of a change, change history, change frequency, and anticipation of a change.
<i>RQ1.3) Where</i> the SPL approaches perform the evolution?	Identify <i>Where</i> the SPL need to evolve according to the taxonomy defined by Buckley <i>et al.</i> (2005). This includes which artifact evolves, its granularity, the impact of a change, and if any kind of change propagation is supported.
<i>RQ1.4) What</i> type of Evolution (static or dynamic) does the approach support?	Identify <i>What</i> type of evolution happens in an SPL according to the taxonomy defined by Buckley <i>et al.</i> (2005). The type of SPL evolution includes availability of the system during the evolution, type of activeness of the system (reactive or proactive), if the approach is open or closed to extensions, the type of safety during the evolution, and other possible types of evolution.
<i>RQ1.5) How</i> SPL approaches support the evolution?	Identify <i>How</i> SPL evolve according to the taxonomy defined by Buckley <i>et al.</i> (2005). This includes the degree of automation, the degree of formalization, and the change type (re-engineering or refactoring).
<i>RQ1.6) What is the SPL life cycle and phase</i> in which the evolution is applied?	Identify what is the SPL life cycle (domain engineering or application engineering) and also which phase (scoping, requirements, architecture, realization, and tests) the approach is applied.
<i>RQ1.7) What is the evaluation procedure</i> from the approach?	Identify what kind of evaluation was performed by applying the approach. The evaluation can be case studies, surveys, controlled experiments, feasibility studies, or not evaluated at all.
<i>RQ1.8) What type of tool support</i> does the approach offer?	Identify whether the approach offers a tool for automation or not.
<i>RQ1.9) What is the context in which</i> the approach is applied?	Identify if the SPL evolution approach is applied within the industry, academia, or both.

Table 4.2: Selected Conferences for Manual Search.

1	International Conference on Software Maintenance and Evolution (ICSME)
2	Software Product Line Conference (SPLC)
3	Conference on Software Analysis, Evolution, and Reengineering (SANER)
4	International Conference on Software Engineering (ICSE)
5	International Workshop on Principles of Software Evolution (IWPSE) / ERCIM Workshop on Software Evolution (EVOL)
6	International Conference on Software Reuse (ICSR)
7	International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS)
8	European Software Engineering Conference (ESEC/FSE)
9	Generative Programming and Component Engineering (GPCE)
10	International Conference/Workshop on Program Comprehension (ICPC)
11	Automated Software Engineering (ASE)
12	European Conference on Software Architecture (ECSA)
13	Working IEEE/IFIP Conference on Software Architecture (WICSA)

Table 4.3: Selected Journals for Manual Search.

1	Transactions on Software Engineering (TSE)
2	Information and Software Technology (IST)
3	Journal of Systems and Software (JSS)
4	Empirical Software Engineering (ESE)
5	Journal of Software: Evolution and Process
6	Communications of the ACM (CACM)
7	Transactions On Software Engineering and Methodology (TOSEM)
8	Software Practice and Experience (SPE)
9	IET Software / IEE Proceedings
10	ACM Sigsoft Software Engineering Notes
11	Journal of Object Technology

part is related to the studies that deal with software evolution and the second one is related to software product lines studies. Table 4.4 shows the search string in which synonyms and alternate terms were joined using the Boolean operator OR; and the Boolean operator AND has been used to join the two parts of the string.

During the search in the digital libraries, the defined search string was applied in the title, abstract and keywords of each paper. Moreover, since each digital library has a specific syntax, the search string syntax was adapted according to each digital library. We also took into account these search terms when the manual search was performed.

Table 4.4: Search String for the Digital Libraries.

```

(((evol*) OR (maint*) OR (chang*) OR (modif*))
  AND
  (“product line”) OR (“product-line”) OR (“product family”) OR (“product-family”) OR
  (“product families”) OR (“product-families”) OR (“family of product”) OR (“families of
  product”) OR (SPL)))

```

This systematic mapping study covers the literature from 1996 up to December 2014. This period was selected because 1996 was the first year in which a conference specifically dedicated to SPL was held (called Workshop on the Design and Evolution of Software Architecture of Product Families, which was the root of the current *Software Product Line Conference - SPLC*).

To validate our search string, we compared the results from the digital libraries with 10 representative studies that should appear in the results. These 10 studies (Acher *et al.*, (S3), Ajila *et al.* (S5), Alves *et al.* (S7), Borba *et al.* (S16), Botterweck *et al.* (S17), Dhungana *et al.* (S30), Lotufo *et al.* (S65), Neves *et al.* (S70), Schulze *et al.* (S95), Thum *et al.* (S104))¹ were previously selected since they are papers with a detailed description of the SPL evolution approach and they also presented clear results after applying the SPL evolution approach. Thus, we have more confidence that the search string was able to find the corrected sample.

4.3.1.3 Selection of primary studies

Each retrieved paper (from the automated search or the manual search) was evaluated by the authors in order to decide whether or not it should be included by considering the title, abstract and keywords. Disagreements in the selection were solved after scanning the whole paper. The studies that met the following inclusion criteria were selected:

- Studies that focus on software product line evolution / maintenance approaches;
- English peer-reviewed studies that provide answers to the research questions; and
- Full Papers with 6 or more pages.

Studies that met at least one of the following exclusion criteria were removed:

- Studies that are not related to the research questions;
- Papers not written in English;
- Introductory papers for special issues;

¹Each one of the references represented as S(*number*) is a paper from the Appendix B.1.

- Books;
- Papers containing extended abstracts; and
- Duplicate reports of the same study in different sources.

4.3.1.4 Quality Assessment

A three-point Likert-scale questionnaire was designed to assess the quality of the selected papers. This questionnaire is composed of three subjective closed-questions and one objective closed-question. We defined the following subjective questions:

- (a) Does the study present a detailed description of the SPL evolution approach?
- (b) Does the study provide guidelines for applying the SPL evolution approach? and
- (c) Does the study present clear results after applying the SPL evolution approach?

The possible answers to these questions were: “Yes, I agree (+1)”; “Partially (0)”; and “No, I do not agree (-1)”. The objective question was:

- (d) Does the study have been cited by other authors? The possible answers to this question were: “Yes (+1)” if the paper has been cited by more than five authors; “Partially (0)” if the paper has been cited from 1 up to 5 authors; and “No (-1)” if the paper has not citation. We used the *Google Scholar*² citations count to rate this question. To not penalize early publications (*i.e.*, papers published in 2014), we rated them as “Partially (0)”.

The quality assessment form (Table 4.5) helped in the identification of good quality papers.

Table 4.5: Quality Assessment Form.

Quality Assessment		+1	0	-1
a.	Does the study present a detailed description of the SPL evolution approach?			
b.	Does the study provide guidelines for applying the SPL evolution approach?			
c.	Does the study present clear results after applying the SPL evolution approach?			
d.	Does the study has been cited by other authors?			

**Adapted from Fernandez et al. (2011)*

²<http://scholar.google.com.br/>

4.3.1.5 Data Extraction Strategy

The employed data extraction strategy was built according to the set of possible answers for each defined sub-research question. By applying the same extraction data criteria to all selected papers, we also facilitated their classification. The data extraction form was created based on the work of [Bosch and Ran \(2000\)](#) for identifying *Why* an SPL evolve, and also it was based on the work of [Buckley et al. \(2005\)](#) to identify *When, Where, What, and How* an SPL evolve. The possible answers to each sub-research question are explained in more detail as follows.

With regard to RQ1.1 (Categories of SPL Requirement Evolution [Why]), an approach can be classified according to the following answers:

- (a) *New product family*: it is the introduction of a new SPL based on an existing one.
- (b) *Introduction of New Product*: Is the introduction of a new product in the SPL.
- (c) *Adding New Features*: it is the introduction of new features into the SPL due to market investigation, new technological opportunities, or competitors.
- (d) *Extend Standard Support*: it is the extension of standard support in the SPL, for example, due to new network communication protocols, component communication standards, or file systems.
- (e) *New Version of Infrastructure*: it is a new version of the infrastructure (*e.g.*, hardware, operating systems) that is used as basis for the SPL products.
- (f) *Improvement of Quality Attribute*: it is the improvement of quality attributes from the SPL and its assets.

With regard to RQ1.2, RQ1.3, RQ1.4, and RQ1.5 the taxonomy proposed by [Buckley et al. \(2005\)](#) was employed in order to classify the evolution within SPL.

Concerning RQ1.2 (Temporal Properties Evaluated [When]), an approach can be classified according to the following:

- (a) *Time of change*: it is the possibility of representing the different phases of the software life-cycle, such as: compile-time, load-time, and run-time. *Compile-time (static)* represents a change made in the software code, thus, the software needs a recompilation to keep running. *Load-time* describes a change while software elements are loaded into an executable system. *Run-time (dynamic)* occurs when the software change is performed during its execution.
 - (b) *Change history*: it is related to the possibility of storing the history of all changes (sequential or parallel) performed in a software. There are tools, called version control tools, that make this change history available.
-

-
- (c) *Change frequency*: changes in a software may be performed continuously, periodically, or at arbitrary intervals. Sometimes, users frequently request changes, but these changes should follow a periodically schedule to be integrated into the software (*i.e.*, during scheduled down-times). Other software systems (*i.e.*, interpreted systems), follow a less-formal change process and may allow developers to integrate changes continuously.
 - (d) *Anticipation*: it refers to the time when the requirements for a software change are foreseen. If the approach supports anticipation of a change it is explicit registered in the extraction form, for example, a design decision to support a future change.

Regarding to RQ1.3 (Object of Change Evaluated [Where]), an approach can be classified according to the following:

- (a) *Artifact*: several types of artifacts may be subject to change over the evolution. We adapted the taxonomy from [Buckley et al. \(2005\)](#) to support the SPL artifacts. A *Core Asset Base* artifact is a group of artifacts from an SPL core. A *Core Asset* artifact is one artifact from an SPL core, apart from the SPL core architecture. An *SPL Architecture* artifact is an artifact from the core SPL architecture. A *Product Architecture* artifact is an artifact from the product architecture. A *Product* artifact is an artifact from the SPL product, apart from the product architecture. Also, an artifact can be classified as *Other* if it does not fit in neither of the previous categories.
- (b) *Granularity*: another influencing factor is the granularity of the change. Within SPL, we defined that *Coarse Grained* artifacts are related to scoping, requirements and architecture. On the other hand, *Fine Grained* artifacts are related to code.
- (c) *Impact*: we also adapted the impact of change from the [Buckley et al. \(2005\)](#) taxonomy to deal with SPL. A *Local* change affects only a product from the SPL. A *Global* change affects the whole SPL.
- (d) *Change propagation*: the *change impact analysis* ([Bohner and Arnold, 1996](#)) tries to measure or assess the consequences of a change. The *traceability analysis* is a way to deal with the change impact analysis, since it establishes explicit relationships between/among two or more artifacts of the software process. In many cases, changes with a global impact require more effort to be performed, which can be estimated by applying effort estimation techniques ([Ramil and Lehman, 2000](#)).

Concerning RQ1.4 (System Properties [What]), an approach can be classified according to the following answers:

- (a) *Availability*: availability indicates whether the SPL has to be continuously available or not during the evolution.
- (b) *Activeness*: the SPL can be *reactive* (changes are driven by an external need) or *proactive* (the SPL autonomously drives changes to itself, related to dynamic SPL).
- (c) *Openness*: SPL are *Open* if they are built to allow extensions. *Closed* SPL do not allow extensions.
- (d) *Safety*: *static safety* SPLs preserve specific safety aspects at compile-time. *Dynamic safety* SPLs are built-in provisions for preventing or restricting undesired behavior at run-time.
- (e) *Other*: if the paper approach deals with some kind property different from the previous cited, it can be classified as *Other* with the name of the new property.

Referring to RQ1.5 (Change Support [How]), an approach can be classified according to the following:

- (a) *Degree of automation*: the degree of automation is strictly related to the tool support. An *Automated* degree supports a fully automated SPL maintenance task. A *Partially* degree supports a partial automated SPL maintenance task. A *Manual* degree does not have a tool to support the SPL evolution task.
- (b) *Degree of formality*: a change support mechanism may be implemented in an *Ad-hoc* way or based on some underlying *Mathematical formalism*.
- (c) *Change type*: *structural changes (Re-engineering)* refer to changes that modify the structure of the software, changing the software behavior. *Semantics (Refactoring)* correspond to the concept of software refactoring, where the behavior of the software still the same over the evolution process.

With regard to RQ1.6 (Phase of the SPL life cycle in which the evolution is applied), an approach can be classified according to the following:

- (a) *Domain Engineering*: when it is related to the SPL core asset, the approach may deal with the SPL core scoping, SPL core requirement, SPL core architecture, SPL core realization, and/or SPL core test.
- (b) *Application Engineering*: when it is related to the SPL products, the approach may deal with the SPL product requirement, SPL product architecture, SPL product realization, and/or SPL product test.

Concerning RQ1.7 (Evaluation Procedure), an approach can be classified according to the following:

-
- (a) *Case study*: if it provides a formal and rigorous study in which data is collected to evaluate the SPL evolution.
 - (b) *Survey*: if it tries to obtain a feedback about the benefits and limitations of the SPL evolution approach (*i.e.*, by applying a questionnaire), over a certain period of time.
 - (c) *Controlled experiment*: if it provides a controlled investigation, which is based on verifying hypotheses concerning the SPL evolution approach.
 - (d) *Feasibility Study*: if it only presents a proof of concept.
 - (e) *Not Evaluated*: if it does not provide any type of evaluation.

Referring to RQ1.8 (Tool Support), an approach can be classified according to:

- (a) *Automatic*: if the approach provides a tool for automatizing the whole/partially SPL evolution.
- (b) *Manual*: if the approach does not provide a tool for automatizing the SPL evolution.

With regard to RQ1.9 (Current Usage), an approach can be classified according to:

- (a) *Academia*: if the approach is applied within the academia context.
- (b) *Industry*: if the approach is applied within the industry context.

A template for data extraction activities was designed to make easier the management of the data extracted for each paper (see Appendix B.2).

4.3.2 Conducting Stage

The defined search string (Table 4.4) was adapted for each electronic database. Each search was performed within the title, abstract, and keywords of the papers (see Appendix B.3).

4.3.2.1 Search from 1996 up to 2014

From those database sources (IEEE *Xplore*, ACM Digital Library, Science Direct, Springer Link), it was found a total of 1,837 papers, as shown in Table 4.6.

From the manual search, a total of 243 papers were found. From this total, 103 were new papers (papers that were not retrieved by the search engines), as shown in Table 4.7 (conferences) and Table 4.8 (journals). The sum of all retrieved papers was 1,940 papers (Table 4.9).

Table 4.6: Total of Retrieved Papers from Databases (1996-2014)

IEEE <i>Xplore</i>	521
ACM Digital Library	308
SpringerLink	489
ScienceDirect	519
Total	1,837

Table 4.7: Total of Retrieved Papers from the Manual Search on Conferences (1996-2014)

International Conference on Software Maintenance and Evolution (ICSME)	1
Software Product Line Conference (SPLC)	46
Conference on Software Analysis, Evolution, and Reengineering (SANER)	4
International Conference on Software Engineering (ICSE)	4
International Workshop on Principles of Software Evolution (IWPSE) / ERCIM Workshop on Software Evolution (EVOL)	2
International Conference on Software Reuse (ICSR)	6
International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS)	10
Generative Programming and Component Engineering (GPCE)	5
Automated Software Engineering (ASE)	1
European Conference on Software Architecture (ECSA)	4
Working IEEE/IFIP Conference on Software Architecture (WICSA)	1
Total	84

Table 4.8: Total of Retrieved Papers from the Manual Search on Journals (1996-2014)

Information and Software Technology (IST)	3
Transactions on Software Engineering (TSE)	1
Journal of Systems and Software (JSS)	5
Journal of Software: Evolution and Process	4
Communications of the ACM (CACM)	3
Transactions On Software Engineering and Methodology (TOSEM)	1
Software: Practice and Experience (SPE)	2
Total	19

Table 4.9: Total of Retrieved Papers (1996-2014)

Database Search*	1,837
Manual Search*	103
Total	1,940

* removing duplicates

4.3.2.2 Selection of Studies from 1996 up to 2014

The selection of the papers was performed by applying 2 filters and extracting the paper information (Figure 4.1), as follows:

- **1st Filter - Applying the Inclusion/Exclusion Criteria (Title and Abstract):** First a brief read of the paper title and abstract was performed. From 1,940 papers, only 463 papers left. Since the number of papers were still high (463), we decided to perform another filter.
- **2nd Filter - Applying the Inclusion/Exclusion Criteria (Abstract, Introduction and Conclusion):** We performed a second filter to ensure that the selected papers are relevant and to perform a more detailed analysis of each paper. It consisted in reading the paper Abstract, Introduction and Conclusion. In parallel with this filter, we also extracted the paper information to fill the data extraction form. Thus, if we decided to select the paper after the 2nd filter, we also extracted its information, according to the data extraction form. After this extraction, we selected another paper, from the remaining papers, to apply once again the 2nd filter. This process was repeated up to the end of the remaining papers. At the end, only 142 papers left.

4.4 Results

The general results, which were revealed by counting the approaches classified in each of the answers of the research sub-questions, is presented in Table 4.10. The complete list of the selected papers from this systematic mapping study is available in B.1. Both the classification of the selected papers in each category and their quality scores are provided in Appendix B.4.

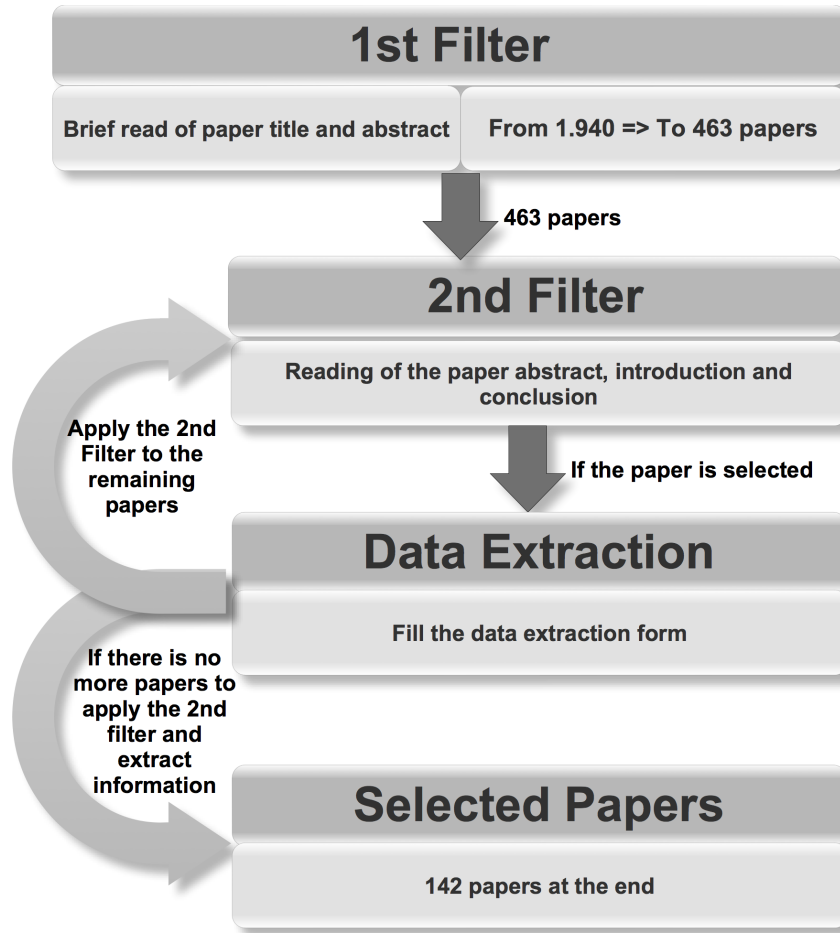


Figure 4.1: Paper Selection Process

Table 4.10: Summary of the Results.

Possible Answers	Results	
	# Studies	Percentage
RQ1.1*		
New Product Family	21	14.79
Intro. of New Product	20	14.08
Adding New Features	103	72.54
Continued on next column		

Continued from previous column

Possible Answers	Results	
	# Studies	Percentage
Extend Standards Support	1	0.70
New Version of Infrastruct.	11	7.75
Improvement of Q.A.	13	9.15

RQ1.2*

Time of Change: Static	102	71.83
Time of Change: Load Time	4	2.82
Time of Change: Run-Time	45	31.69
Change History: Parallel	14	9.86
Change History: Sequential	5	3.52
Change Freq.: Arbitrary	10	7.04
Change Freq.: Continuous	0	0.00
Change Freq.: Periodically	3	2.11
Anticipation	10	7.04

RQ1.3*

Artifact: Core Asset Base	40	28.17
Artifact: Core Asset	82	57.75
Artifact: SPL Architecture	26	18.31
Artifact: Prod. Architecture	10	7.04
Artifact: Product	21	14.79
Artifact: Other	1	0.70
Granularity: Fine Grained	46	32.39
Granularity: Coarse Grained	127	89.44
Impact: Local	30	21.13
Impact: Global	138	97.18
Change Prop.: Impact	22	15.49
Change Prop.: Effort Est.	3	2.11
Change Prop.: Traceability	6	4.23

Continued on next column

Continued from previous column		
Possible Answers	Results	
	# Studies	Percentage
RQ1.4*		
System not Available	106	74.65
System Available	40	28.17
Activeness: Reactive	103	72.54
Activeness: Proactive	43	30.28
Openness: Closed	0	0.00
Openness: Open	6	4.23
Safety: Static	2	1.41
Safety: Dynamic	4	2.82
RQ1.5*		
Automation: Manual	58	40.85
Automation: Partially	43	30.28
Automation: Automated	41	28.87
Formality: Ad hoc	96	67.61
Formality: Formal	46	32.39
Change Type: Structural	42	29.58
Change Type: Semantics	50	35.21
RQ1.6*		
Domain Eng.: Scoping	33	23.24
Domain Eng.: Requirement	100	70.42
Domain Eng.: Architecture	64	45.07
Domain Eng.: Realization	54	38.03
Domain Eng.: Test	7	4.93
App. Eng.: Requirement	25	17.61
App. Eng.: Architecture	34	23.94
App. Eng.: Realization	16	11.27
App. Eng.: Test	4	2.82
Continued on next column		

Continued from previous column		
Possible Answers	Results	
	# Studies	Percentage
RQ1.7		
Not Evaluated	14	9.86
Feasibility Study	121	85.21
Controlled Experiments	2	1.41
Case Studies	4	2.82
Surveys	0	0.00
Other	1	0.70
RQ1.8		
Automatic	65	45.77
Manual	77	54.23
RQ1.9*		
Academia	133	93.66
Industry	12	8.45

*These Sub-RQs are not exclusive. A paper can be classified in one or more of the answers. Thus, the sum of the percentages is over 100%.

The following sub-sections present the analysis of the results for each sub-research question.

4.4.1 Why SPL approaches need to deal with evolution? (RQ1.1)

The results for RQ1.1 revealed that around 15% of the papers deal with the evolution to build a new product family (see Table 4.10). For instance, we found representative examples of these approaches within the work from Acher *et al.* (S3), Kang *et al.* (S51) and Lopez-Herrejon *et al.* (S131).

Acher *et al.* (S3) presented a comprehensive tool supported process for reverse engineering architectural feature models. The tool can, in an automatic way, extract and combine different variabilities of an architecture. However, the final document needs to be validated with a document generated by a software architect.

Kang *et al.* (S51) described a re-engineering method for transforming, through feature-oriented method, home robot applications into SPL. According to them, a feature-oriented

re-engineering approach may help robot builders to achieve some of the SPL benefits (*i.e.*, decrease in development cost and increase in application flexibility). They start with a legacy system, extract its architecture, and then extract the features. Following, they refine the feature model according to the SPL and generate the SPL architecture/components.

Lopez-Herrejon *et al.* (S131) propose to apply reverse engineering in a set of system variants to achieve an SPL. Their approach is iterative where all the involved stakeholders go through multiple iterations and feature models, SPL architecture, and supporting platform are successively refined. Thus, at the end of their approach they provide a first working feature model.

Also, the results for sub-research question 1.1 identified that around 14% of the papers (see Table 4.10) deal with evolution for introducing a new product into the SPL, for instance: Ajila and Kaba (S4) and Borba *et al.* (S16).

Ajila and Kaba (S4) have presented some mechanisms to support the evolution of a software product line in order to support a new product. According to them, traceability mechanisms are essential to perform this evolution. Thus, to evaluate the impact of a change when adding a new product in an existing SPL, they proposed operations and impact analysis functions.

Borba *et al.* (S16) proposed a theory for refactoring products into an SPL. However, the refactoring that they consider is more than a just refactoring, is a refactoring considering quality. Thus, they call it as a refinement. The main idea is to evolve an SPL by improving the design or by adding new products, however, keeping the existing ones. This evolution is based on formal language, proofs, theorems and lemmas.

The majority of the papers deal with evolution in SPL by adding new features (72.54%, as is shown in Table 4.10). For instance, Alves *et al.* (S7), Botterweck *et al.* (S17) and Quinton *et al.* (S135).

Alves *et al.* (S7) extended the traditional notion of program refactorings for SPL, in which feature models are refactored. They focused on feature model refactoring improving (maintaining or increasing) the set of all possible configurations of the initial feature model. Thus, they propose a catalog of feature model refactorings, allowing an increase within the configurations.

Botterweck *et al.* (S17) presented an approach for extending model-driven product line engineering to an automated technique for SPL evolution. Their approach deals with the feature model variability over time and proposes a “*Catalogue of evolution operators*”.

Quinton *et al.* (S135) discuss edits in the feature model taking into account the cardinalities. Thus, they proposed a formal approach to explain inconsistencies within cardinality-based feature models. They claim that a tool is necessary to support the detection of inconsistencies

after adding, removing, or changing the feature model.

Only Axelsson's work (S11) deals with the evolution of an SPL to extend standards support (less than 1%, see Table 4.10).

Axelsson (S11) investigated changes applied into an existing SPL architecture, taking into account what caused the change, quality attributes, and technical aspects.

Around 7.7% of the papers deal with evolution in SPL due to a new version of the infrastructure (see Table 4.10). Examples are the work from Bencomo *et al.* (S15) and Weyns and Michalik (S109).

The approach proposed by Bencomo *et al.* (S15) provides variability management by systematically promoting software reuse and by using models. In their approach, they allow the specification of structure and behaviour from a Dynamically Adaptive Systems (DAS).

Weyns and Michalik (S109) developed an automated approach to evolve deployed SPL products. They proposed an architecture meta-model to store the necessary knowledge to evolve an SPL and an algorithm to generate tasks that should be performed to evolve the SPL based on the meta-model knowledge.

Around 9% of the papers evolve an SPL due to an improvement of quality attribute (see Table 4.10). For example, Neves *et al.* (S70) and Schulze *et al.* (S95).

Neves *et al.* (S70) identified and described precisely a number of SPL safe evolution templates. The templates include: split an asset; refine an asset; add a new optional feature; add a new mandatory feature; replace a feature expression; add a new alternative feature; add a new OR feature; and delete an asset. The proposed templates are based on refactoring an SPL, where the behavior of the SPL remains the same after the evolution.

Schulze *et al.* (S95) proposed an extension of existing refactoring techniques for Feature-Oriented Programming (FOP). They provide exemplary refactorings for feature-oriented SPL in a catalogue-like manner. Additionally, they discuss the generalizability of their definition and the actual refactoring towards annotative SPL implementation techniques.

4.4.2 When the SPL approaches perform the evolution? (RQ1.2)

The results for the sub-research question 1.2 shown that around 72% of the papers deal with static SPL evolution (see Table 4.10). For instance, we found representative examples of these approaches in the work from Anastasopoulos (S9), Thüm *et al.* (S104), and Acher *et al.* (S119).

The research question studied by Anastasopoulos (S9) addresses how configuration management can be successfully used for managing the evolution in an SPL. Hence, the contribution of this work lies in the reduction of steps SPL engineers have to perform in order to achieve

evolution control when dealing with branches in an SPL. These steps are performed in an static way.

Thüm *et al.* (S104) presented and evaluated an algorithm to determine, in an static manner, the relationship between two feature models (*i.e.*, specialization/refactoring/generalization) using satisfiability solvers.

Acher *et al.* (S119) presented a practical support for synthesising a Feature Model (FM) from a set of configurations. They propose a tool support for performing static operations in FMs, such as, reverse engineering, refactoring, merging, and slicing.

Around 3% of the papers deal with load time evolution in SPL (see Table 4.10). An example is the approach presented by Hallsteinsen *et al.* (S44).

Hallsteinsen *et al.* (S44) proposed to adapt an SPL during compile time (launched system) and also during run-time. The proposed adaptive systems are built with explicitly variability being part of the architecture. In this way, they are able to relieve the developer from much of the added complexity of adaptivity.

Run-time (dynamic) evolution in SPL corresponds to 31.69% of the papers (see Table 4.10). For instance, the work from Cetina *et al.* (S21), Rosenmuller *et al.* (S85), and Cetina *et al.* (S123).

Cetina *et al.* (S21) identified and addressed two challenges related to human subjects in Dynamic SPL (DSPL). The first one is to allow DSPL users to trigger the run-time reconfigurations, and the second one is to understand the reconfigurations effects. They provided some guidelines for DSPL to: introduce user confirmations to reconfigurations; improve reconfiguration feedback; and introduce rollback capabilities to reconfigurations.

The approach presented by Rosenmuller *et al.* (S85) allows to switch the feature binding time through the same code base. Their approach statically allows the generation of dynamic bindings, which reduces the overhead. Moreover, the approach also provides composition safety of dynamic binding by using a transformed feature model. Thus, they provide transformation rules for creating dynamic binding units.

Cetina *et al.* (S123) presented a DSPL case study in which they used a Smart Hotel for performing run-time reconfigurations. They used real devices and human subjects to understand the reliability-based risk of reconfigurations. Thus, they focus on two attributes of reliability-based risk: probability of malfunctioning (Availability) and the consequences of malfunctioning (Severity).

Around 10% of the papers deal with parallel evolution in SPL (see Table 4.10). For instance, the work from Thao *et al.* (S103).

The tool proposed by Thao *et al.* (S103), called MoSPL, is a component-based Software

Configuration Management (SCM) system, which deals with product versioning, product derivation, and the version management of an evolving SPL. MoSPL explicitly manages logical constraints and derivation relationships among the core assets and products.

Sequential evolution in SPL represent less than 3.6% of the papers (see Table 4.10). An example is the work from Chen *et al.* (S23).

Chen *et al.* (S23) presented a discrete event simulation which is used to provide a framework for the simulation of SPL. This simulation can execute in alternative settings (*i.e.*, sequential). They have created an environment to make easy the strategic management and the long-term forecasting, according to the SPL development and evolution.

The results also shown that around 7% of the papers deal with arbitrary evolution in SPL (see Table 4.10). For instance, the approaches presented by Anastasopoulos *et al.* (S10), Anquetil *et al.* (S102), and Dintzner *et al.* (S124).

The solution presented by Anastasopoulos *et al.* (S10) is based on configuration management, which is an established management discipline for controlling the evolution of software systems. Since traditional configuration management does not address special aspects pertaining to product lines, a set of extensions are proposed to fill the respective gaps. The solution is made up of two components: RIPLE-EM and PULSE. The study tries to bring them both together to solve the problem with the management activity in evolving SPL.

Anquetil *et al.* (S102) proposed the AMPLE Traceability Framework (ATF). This framework deals with SPL traceability issues and can be customized according to the SPL context. They offer support for versioning (version/time links) using SVN in eclipse.

Dintzner *et al.* (S124) presented a classification of feature changes that occur in the Linux kernel feature model based on the Kconfig language and a corresponding tool, called FMDiff, to extract them. Their classification describes feature changes on several levels of granularity.

None approach deals with continuous evolution within SPL (see Table 4.10).

Around 2.1% of the approaches deal with periodically SPL evolution (see Table 4.10). For instance, the approach from Lotufo *et al.* (S65).

Lotufo *et al.* (S65) identified some categories for changing the Linux model (called reasons for edits): new functionality; retiring obsolete features; clean-up/maintainability; support to changes in C code; build fix; and change variability. These edits are performed periodically within the Linux model.

Finally, around 7% of the approaches deal with anticipation in SPL evolution (see Table 4.10). Two examples are the approaches presented by Simon *et al.* (S100) and the approach from White *et al.* (S141).

Simon *et al.* (S100) proposed a lightweight iterative process to re-engineering a legacy

system into SPL. This process has some guidelines on how to perform this re-engineering, including the feature anticipation (where a market analysis is performed to anticipate possible new features).

White *et al.* (S141) have proposed an automated approach for deriving configurations according to a set of requirements. They called their technique the Multi-step Software Configuration problem Solver (MUSCLES). MUSCLES uses Constraint Satisfaction Problems (CSPs) to foresee changes in the next versions of the feature model.

4.4.3 Where the SPL approaches perform the evolution? (RQ1.3)

The results for RQ1.3 revealed that around 28.1% of the papers deal with the evolution within the whole core asset base (see Table 4.10). For instance, we found representative examples of these approaches within the work from Dhungana *et al.* (S29), Inoki and Fukazawa (S49), and Passos *et al.* (S134).

Dhungana *et al.* (S29) have developed a model-based approach that allows the SPL definition, management, and utilization. They assume that is easier to maintain a small model than a larger one. Thus, their approach proposes to create model fragments, which describes the variability of the selected parts of the system, instead of a large model.

Inoki and Fukazawa (S49) proposed an approach for evolving core assets evolution based on the *kaizen* approach. They defined kaizen patterns (patterns for evolving features and assets from a core asset), composed of the SPL knowledge. The approach also allows the improvement of current work standard.

Passos *et al.* (S134) inspected over 500 Linux kernel commits (around four years of development) to get insights of how variability models and their related artifacts coevolve. Thus, they identified a catalog of evolution patterns based on the Linux kernel variability model and its related artifacts (Makefiles and source code).

Around 57.7% of the approaches deal with evolution within a core asset (see Table 4.10). Examples are the approaches from Lee and Muthig (S60) and Savolainen and Kuusela (S90).

Lee and Muthig (S60) proposed an approach to perform the analysis and the specification of features that vary as a part of reconfigurations at run-time. They focused on providing a formal base that can be used as a basis for adding other new definitions and consistency rules.

The approach proposed by Savolainen and Kuusela (S90) deals with SPL specifications. It is based on the definition hierarchy method - a model-based requirements engineering technique. They consider the evolution during the requirements specification. Each requirement has an entry on a table, and it is associated to products through a prioritization scheme. They call this

analysis volatility analysis.

The results shown that around 18.3% of the papers deal with SPL architecture (see Table 4.10). For instance, the approaches from Diaz *et al.* (S31), Gomaa and Hussein (S40), and Gamez and Fuentes (S125).

The approach from Diaz *et al.* (S31) addresses change impact analysis when evolving a Product Line Architecture (PLA). They propose to use traceability links and propagation rules within PLA. Thus, since there is a need to handle the variability within the traceability and the PLA, their approach supports the specification of the PLA variability, knowledge, and also it allows to trace the variability between requirements and PLAs.

Gomaa and Hussein (S40) state that for each software architecture pattern, there must be a reconfiguration pattern, which allows to adapt the architecture dynamically. This reconfiguration pattern uses a state machine and orthogonal state-charts. Thus, they proposed a change management model to describe this reconfiguration.

Gamez and Fuentes (S125) presented an SPL evolution approach to deal with the FamiWare, which is *a family of middleware for intelligence environments*. In this approach, changes performed in the feature model are automated and propagated to the architectural components of the middleware. Moreover, the approach also calculates the effort when performing a change.

Around 7% of the approaches deal with evolution within the product architecture (see Table 4.10). For example, the approaches in Ajila and Kaba (S5) and Weyns and Michalik (S110).

Ajila and Kaba (S5) presented basic mechanisms to support SPL process evolution. These mechanisms share four strategies: change identification, change impact, change propagation, and change validation. This study also examines three kinds of evolution processes: architecture, product line, and product.

The research presented by Weyns and Michalik (S110) is concerned to update one or more deployed products of an SPL. Their particular focus is on updating SPL products that require online updates with minimal interruption. Thus, they defined a framework for updating SPL products.

Results revealed that around 14.7% of the approaches deal with product evolution (see Table 4.10). For instance, the approaches proposed by Heider *et al.* (S47) and Romanovsky *et al.* (S84).

Heider *et al.* (S47) presented an approach supported by a tool called VaMoRT (Variability Modeling Regression Testing). It supports impact analyses during variability modeling by determining the impact of changes on existing products.

The approach proposed by Romanovsky *et al.* (S84) has the aim of documenting SPL documents using a button-up approach (starting from the documentation of one product and

identifying reused documentation from the following products). They proposed a tool that uses XML for documenting SPL.

Less than 1% of the approaches deal with evolution of other SPL artifact (see Table 4.10). An example is the approach proposed by Chen *et al.* (S23).

To improve the SPL evolution (and also development), Chen *et al.* (S23) proposed a simulation tool called DEVSJAVA and a life cycle cost estimation model (COMPLIMO).

Around 32.3% of the approaches deal with evolution of fine grained SPL artifacts (see Table 4.10). For instance, the approaches presented in Alves *et al.* (S8), Lopez-Herrejon *et al.* (S64), and Kanda *et al.* (S128).

Alves *et al.* (S8) addressed the issues of structuring and evolving SPL in highly variant domains. They proposed an approach that combines extractive and reactive SPL techniques to extract the SPL variation from a current software, and adapt the new SPL to support other products. Their approach uses refactorings derived from simple Aspect-Oriented Programming (AOP) laws.

Lopez-Herrejon *et al.* (S64) identified eight refactoring patterns. According to them, these patterns allow the extraction of code fragments that implement features.

Kanda *et al.* (S128) proposed to extract the evolution history of SPL products based on the code. Their approach relies only on the source code, thus, it does not consider names, numbers, or release dates. They proposed a *Product Evolution Tree* in which each node of the tree is a product, the edges connect similar products, and the labels show the product similarity and the evolution direction.

Results revealed that around 89.4% of the approaches deal with evolution of coarse grained SPL artifacts (see Table 4.10). Examples are the approaches proposed by Deelstra *et al.* (S28), Seidl *et al.* (S96), and Acher *et al.* (S120).

Deelstra *et al.* (S28) identified whether, when, and how variability in SPL should evolve (COSVAM assessment method). The reasons are associated to a number of methodological and knowledge issues. Methodological issues include: unstructured; reactive instead of proactive; generalized instead of optimal decisions; lack of removing obsolete variability; and addressing only one layer of abstraction. Knowledge issues include: implicit variability; neglecting implementation dependencies; and insufficient number of alternative solutions.

Seidl *et al.* (S96) firstly presented a classification for evolutions which stores the effects of changes performed within the feature model. Secondly, they presented a conceptual basis for co-evolving models and the feature model.

Acher *et al.* (S120) presented a tool-supported process for reverse engineering and evolving architectural feature models. They developed automated techniques to extract and combine

different variability descriptions of a software architecture.

Around 21.1% of the approaches have a local impact during the SPL evolution (see Table 4.10), such as the approaches presented by Abbas *et al.* (S2) and Wu *et al.* (S113).

Abbas *et al.* (S2) proposed the Autonomic Software Product Lines (ASPL), where products have online learning mechanism that continuously evolve a product's internal knowledge and, in addition, may trigger internal adaptations.

The study presented by Wu *et al.* (S113) examined the evolution in requirements and architecture of 10 products, including 51 major releases. They address the highly complex inter-relationship between SPL requirements, PLA, and product architectures. From the architectural impact, they identified some requirement changes, such as: default requirement changes; urgent requirement changes; tentative requirement changes; customer-specific requirement changes; large scale requirement changes; and internal maintainability requirement changes. Moreover, it was identified six typical architectural evolution patterns, namely linear evolution, clone, derivation, merge, synchronization, and propagation.

The majority of the approaches, around 97.1%, have a global impact during the SPL evolution (see Table 4.10). Examples of these approaches are the work from Cordy *et al.* (S24), Knodel *et al.* (S56), and Benlarabi (S122).

Cordy *et al.* (S24) identified two special classes of features, the conservative and the regulative features. They propose a formal method to define if a feature is conservative or regulative in a given SPL. Thus, they formally define interesting classes of features, which allow to model check only a subset of the products when a new feature is added.

The study presented by Knodel *et al.* (S56) proposed to integrate existing assets into the SPL core asset and also it relates reverse engineering techniques to recover assets from documents and history analysis.

Benlarabi's (S122) approach consists on building a co-evolution model using biological co-evolution techniques, specifically Cladistics classification. This technique was extensively used in biology and it demonstrated its efficiency in biological co-evolution analysis. The approach started by collecting data from historical information about a SPL evolution, then these data is saved into a subversion system with clear comments on the purpose of the change of the whole SPL.

Results revealed that around 15.4% of the approaches deal with change impact analysis (see Table 4.10). Examples are the approaches presented by Peng *et al.* (S77), Tizzei *et al.* (S105), and Gaia *et al.* (S126).

The main contribution from Peng *et al.* (S77) work is threefold. First, they proposed to analyze variability evolution from the prospect of context and requirement changes and presented

a set of evolution rules that relate changes of contexts and requirements to those of feature variations. Second, they developed an algorithm to analyze the impact scope to classify context or requirement changes in the variability evolution. Finally, they illustrated a systematic process using these rules and the impact analysis algorithm to help identify possible changes in contexts and requirements and rank their influences on features.

The study presented by Tizzei *et al.* (S105) proposed techniques for PLA stabilization. According to them, the design stability can be achieved by combining components and aspects. This finding was based on the analysis of change impact and modularity.

Gaia *et al.*'s (S126) analysis on SPL evolution refers to interpretation of collected measures related to stability and modularity. The change impact metrics were used for the analysis of stability and separation of concern metrics were used by modularity.

Around 2.1% of the approaches deal with effort estimation (see Table 4.10). For instance, the approach present in Chen *et al.* (S23). This approach is able to represent volatility in multiple levels and has capacity to tie the volatility estimation to one SPL member specification.

Around 5% of the approaches have traceability analysis (see Table 4.10). Examples are the approaches proposed by Heider *et al.* (S41) and Guo *et al.* (S43).

Heider *et al.* (S41) presented the PUPLE (Product Updates in Product Line Engineering) approach and discussed its implementation. PUPLE tries to update products in an automated way by solving conflicts and it also helps users on manual conflict resolution when the automatic update does not work.

The study presented by Guo *et al.* (S43) formalizes feature models by defining its primitive elements, the syntactical and semantic consistency constraints as the well-formedness rules. From this formalization, it is obtained a set of primitive operations which can represent any modification of a feature model. They apply and extend techniques from ontology evolution to propose a systematic approach to consistency maintenance for evolving feature models.

4.4.4 What type of Evolution (static or dynamic) does the approach support? (RQ1.4)

The results for sub-research question 1.4 revealed that around 74.6% of the approaches evolve static SPL, which products are not always available during the evolution (see Table 4.10). For instance, the approaches described by Kim *et al.* (S53) Smith *et al.* (S101), and Krishnan *et al.* (S130).

Kim *et al.* (S53) described their approach as a mix of forward and reverse engineering. Their approach has principles and guidelines for designing an SPL platform and was evaluated within

the digital audio and video domain.

The study presented by Smith *et al.* (S101) proposes Options Analysis for Re-engineering (OAR) for SPL, which is a method that provides a systematic approach to make decisions on the mining of components from legacy systems for use in SPL. They show some tasks for selecting the existing components and re-engineering them into SPL.

Krishnan *et al.* (S130) investigated whether classification-based prediction of failure-prone files improves as an statical product line evolves. They based their investigation on 4 Eclipse products. According to them, the SPL matures and the learner performance improves significantly.

On the other side, around 28.1% of the approaches evolve an SPL that is always available (see Table 4.10). Examples are the approaches from Gomaa and Hashimoto (S38), and Adelsberger *et al.* (S121).

The study presented by Gomaa and Hashimoto (S38) addresses a dynamic SPL for Service Oriented Architecture (SOA). They extended the PLUS method's to support the dynamic adaptation.

Adelsberger *et al.* (S121) proposed an approach to analyze the migration of objects within Feature-Oriented Product lines (FOP), where an live object may move from one behaviour configuration to another. They evaluated their approach using 9 SPLs from the Fuji repository.

The results revealed that 72.54% of the approaches are reactive (see Table 4.10), such as the approaches from Loesch and Ploedereder (S62), and Schulze *et al.* (S137).

Loesch and Ploedereder (S62) proposed an approach that analyzes the SPL variability and presents what is the real usage of the features within the SPL products. The method also looks for unused features and remove them.

Schulze *et al.* (S137) presented a precondition-based approach for implementing variant-preserving refactoring for feature-oriented SPLs. Furthermore, they provided details about the implementation of their refactoring tool called VAmPiRE and how they realized decomposition and reuse of refactorings. Finally, they only implemented one concrete refactoring (Pull Up Method), thus, they considered that they provided a starting point for the implementation of further refactorings with their tool.

Around 30.2% of the approaches are proactive (see Table 4.10). Examples are the approaches proposed by Cetina *et al.* (S22), Pleuss *et al.* (S80), and Koscielny *et al.* (S129).

Cetina *et al.* (S22) proposed mixed DSPL as an intermediate solution that takes the benefits of both connected and disconnected DSPL architectures.

The study shown by Pleuss *et al.* (S80) presents a model-driven approach to handle the evolution of SPL on feature model level. Their work is geared towards the following goals:

documentation (of previous evolution); planning (of future evolution); abstraction (fragments); automation; and analysis.

Koscielny *et al.* (S129) proposed a prototypical implementation of DELTAJ 1.5, which supports JAVA object-oriented features. Furthermore, they improve the specification of the product line declaration by providing a separate language.

According to the results, none approach was identified as closed (see Table 4.10).

The results reveal that around 4.2% of the approaches are open (see Table 4.10), such as the approach presented by Wolfinger *et al.* (S111). They demonstrated the integration of product line engineering and plug-in techniques. Together with their industrial partner they have identified several usage scenarios for run-time adaptation in the Enterprise Resource Planning (ERP) domain confirming the need of such an approach. The integrated approach allows the adaptation of the system by the user according to the variability model.

Less than 2% of the approaches deal with static safety in SPL evolution (see Table 4.10). An example is the approach from Liu *et al.* (S61).

Liu *et al.* (S61) developed a tool support in order to safe evolve SPL requirements through a model-based approach. They use fault tree analysis to identify if a new requirement can be safely included into the SPL.

Around 2.8% of the approaches deal with dynamic safety in SPL evolution (see Table 4.10), such as the approach revealed by Rosenmuller *et al.* (S86).

Rosenmuller *et al.* (S86) bridged the gap between feature-based variability modeling and component-based run-time adaptation, by integrating generative SPL engineering and DSPL.

4.4.5 How SPL approaches support the evolution? (RQ1.5)

The results for RQ1.5 shown that around 40.8% of the revealed approaches are manual (see Table 4.10). For instance, the approach presented by Niu *et al.* (S71).

Niu *et al.* (S71) dealt with the evolution and inconsistency of viewpoints from feature models. They proposed several viewpoints (business user viewpoint, secure usage viewpoint, HW platform viewpoint) from the feature model and evolve them in a manual way.

Around 30.2% of the approaches are partially automated (see Table 4.10), such as the approaches revealed by Valente *et al.* (S108) and Muschevici *et al.* (S133).

To speed up product line extraction, Valente *et al.* (S108) described a semi-automatic approach which allows to annotate the code from optional features. This approach was based on a tool for SPL development called Colored IDE (CIDE).

Muschevici *et al.* (S133) introduced a support for modelling dynamic variability in the

Abstract Behavioural Specification language (ABS). They designed an adaptive run-time environment and an ABS compiler which generates Java code. These tools are implemented and available as part of the proposed ABS tool framework.

The results reveal that around 28.8% of the approaches are automated (see Table 4.10). Examples are the approaches from Ribeiro *et al.* (S82) and Schulze *et al.* (S138).

Ribeiro *et al.* (S82) proposed the idea of emergent interfaces. The idea is to capture dependencies between the feature a programmer is maintaining and the others. These interfaces emerge and give information about other features they might impact with their current maintenance task.

Schulze *et al.* (S138) provide a catalogue of 23 refactorings for Delta Oriented Programming (DOP) which supports the efficient evolution of SPLs. They provide a tool support for most of their proposed refactorings as an Eclipse Plugin.

Most of the approaches (67.6%) are applied in an *ad-hoc* manner (see Table 4.10), such as the approach revealed according to Seidl and Aßmann (S139). They presented a meta-model, which allows the creation of models including software ecosystem elements and their relationships. Moreover, this meta-model represents the changes over the time, however, they do not applied any math formalisms.

Around 32.3% of the approaches have math formalisms (see Table 4.10). For instance, the approach from Damiani and Schaefer (S27). They presented the Dynamic Delta-Oriented Programming, an extension of DOP, as a flexible approach to realize dynamic SPL. Based on math formalisms, it is possible to include/evolve a product of an SPL dynamically by adding the code of the product in the code base, change the SPL declaration and the automaton.

The results reveal that around 29.5% of the approaches focus on re-engineering-structural (see Table 4.10), such as the approaches from Bayer *et al.* (S13) and Wu *et al.* (S112).

Bayer *et al.* (S13) based on wrapping Fortran code to achieve (re-engineering) C++ code. They proposed the RE-PLACE approach. RE-PLACE stands for Re-engineering-Enabled Product Line Architecture Creation and Evolution. Based on existing assets (components) they build (or wrap) them into an SPL. They also use a knowledge base (with the documents, *e.g.*, existing features, task scenarios, reusable components, and so on). Every change is confirmed with a specialist.

Wu *et al.* (S112) proposed an semi-automatic approach to recovery an SPL based on legacy products. Their approach considers a mapping among design elements from the different products and the identification of the variability among the elements.

Around 35.2% of the approaches focus on refactoring-semantics (see Table 4.10). Examples are the approaches revealed by Hanssen *et al.* (S45) and Patzke *et al.* (S76).

Hanssen *et al.* (S45) presented some problems that agile practitioners usually deal in SPL

long-term evolution. They also have shown how agile methods are affected by this software entropy and proposed a combination of two strategies to address this issue.

The method presented by Patzke *et al.* (S76) is the PuLSETM-E (ProdUct Line Software Engineering - Evolution), which deals with the evolution of SPL assets (infrastructure code).

4.4.6 What is the SPL life cycle and phase in which the evolution is applied? (RQ1.6)

The results for sub-research question 1.6 revealed that around 23.2% of the approaches are applied within the domain engineering scoping (see Table 4.10), such as the approaches from Elsner *et al.* (S33) and Seidl *et al.* (S140).

Elsner *et al.* (S33) approach deal with the “variability in space” and the “variability in time”. To evolve an SPL, they proposed the following steps: proactive planning (scoping); tracking; analysis; correction; and realignment.

According to Seidl *et al.* (S140), feature models do not capture the variability in time (evolution), making impossible to deal with versions of variable assets. Thus, they deal with feature models and propose Hyper Feature Models explicitly providing feature versions as configurable units for product definition.

Around 70.4% of the approaches addressed domain requirements engineering (see Table 4.10). For instance, the approaches revealed by Kim *et al.* (S55) and Murguzur *et al.* (S132).

Kim *et al.* (S55) focused on dynamic change of quality requirements in an SPL. They propose a goal and scenario driven approach, which treats dynamic quality requirements as goals to be achieved in an SPL, and analyzes dynamic variabilities to meet the goals through scenarios.

Murguzur *et al.* (S132) approach deals with context variability modeling for DSPLs. They claim that their approach provides a way to anticipate changes when a new context feature is required. They evaluated the approach through a wind farm use case.

The results reveal that around 45% of the approaches deal with evolution with domain engineering architecture (see Table 4.10). Examples are the approaches by Gomaa and Hussein (S39) and Trinidad *et al.* (S106).

Gomaa and Hussein (S39) proposed an approach relating software reconfiguration pattern and SPL architecture patterns. Their approach reconfigures, in an automatic way, an SPL from one state to another. They claim that the approach promotes software evolution because the SPL could be modified after it becomes operational and then dynamically reconfigured.

The objective of Trinidad *et al.* (S106) is to generate a component architecture that supports the dynamics of products and which is easily inferred from a feature model.

Around 38% of the approaches addressed the realization in the domain engineering (see Table 4.10), such as the approach revealed by Kästner *et al.* (S52).

In summary, Kästner *et al.* (S52) have presented a formal model for a programming language called LJAR, which supports virtual (using `#ifdef` or CIDE) and physical (using AHEAD or FeatureHouse) separation of features. They also have implemented refactorings in CIDE.

The results reveal that around 4.9% of the approaches deal with test evolution for the domain engineering (see Table 4.10). For instance, the approach from Abbas *et al.* (S1). This approach provides an automatic support to reconfigure products. If new variants are added or existing ones removed, they will be tested.

Around 17.6% of the approaches tackled requirements in the application engineering (see Table 4.10). For instance, the approach revealed by Wu *et al.* (S113). They examined the collection of requirements changes in 10 member products and 51 major releases. They identified several requirement changes from the point of view of architectural impact, such as: default requirement changes; urgent requirement changes; tentative requirement changes; customer-specific requirement changes; large scale requirement changes; and internal maintainability requirement changes.

The results reveal that around 23.9% of the approaches deal with architecture evolution for the application engineering (see Table 4.10), such as the approach from Michalik *et al.* (S68). They have developed an architecture-centric approach to support SPL products updates. Their approach supports the definition of viewpoints, to capture the stakeholders' concerns, and they also proposed a tool support to assist the architecture reconstruction.

Around 11.2% of the approaches investigated realization in the application engineering (see Table 4.10). For instance, the approach revealed by Mende *et al.* (S67). They proposed a grow-and-prune model to identify similarities between two product functions. Based on this model, SPL products refactoring can be performed.

The results reveal that around 2.8% of the approaches deal with test evolution for the application engineering (see Table 4.10). An example is the approach from Ajila *et al.* (S5). Besides dealing with evolution of product architecture artifacts, Ajila *et al.* (S5) also deals with regression testing to validate a change in the SPL product.

4.4.7 What is the evaluation procedure from the approach? (RQ1.7)

The results for RQ1.7 revealed that around 9.8% of the approaches were not evaluated (see Table 4.10), such as the approach from Perrouin *et al.* (S78). They presented their contribution, which is a modeling process supporting the definition of variability spaces, however, none evaluation was conducted.

Most of the approaches, 85.2%, were evaluated through a feasibility study (see Table 4.10). For instance, the approaches by Heider *et al.* (S46) and Montero *et al.* (S69).

The approach proposed by Heider *et al.* (S46) uses simulation to understand the SPL evolution (model maintenance effort and model complexity). However, they evaluated their approach through what they called simulation experiment.

Since the focus from Montero *et al.* (S69) is on Business-Driven Development and SPL, they deal with run-time evolution of these kind of business systems. They call this proposal Business Family Engineering. As most of the papers that fit in this answer (Feasibility study), they said that they evaluated their approach through a case study. However, they do not follow a systematic guide to conduct a case study as proposed by Runeson and Höst (2009). Thus, these studies were considered feasibility studies.

Less than 2% of the approaches are evaluated through controlled experiments (see Table 4.10), such as the approach from Michalik *et al.* (S68). They performed a controlled experiment to evaluate the effectiveness of the propose architecture-centric approach.

Around 2.8% of the approaches are evaluated using case studies (see Table 4.10). For instance, the approaches revealed by Heider *et al.* (S41) and Hellebrand *et al.* (S127).

Heider *et al.* (S41) conducted a case study to evaluate PUPLE. The aim of this case study was to understand the impact of changes within the variability model (decision-oriented). Moreover, they also analyzed the degree in which PUPLE supports updating partially or completely the derived products.

Hellebrand *et al.* (S127) performed a case study to investigate the co-evolution of variability models and the source code, based on a set of derived metrics and methods.

None approach was evaluated using surveys (see Table 4.10).

Less than 1% of the approaches used other type of evaluation (see Table 4.10). For instance, the approach from Anastasopoulos (S9). The approach defined by Anastasopoulos was evaluated through a quasi-controlled experiment and by a simulation study.

4.4.8 What type of tool support does the approach offer? (RQ1.8)

The results for sub-research question 1.8 revealed that around 45.7% of the approaches are automatic (see Table 4.10). For instance, the approaches from Nunes *et al.* (S72) and Romero *et al.* (S136).

Nunes *et al.* (S72) proposed an approach to check the evolution history of each product and try to identify the common and variable features among those evolutions. The automation is performed by a tool called RecFeat.

Romero *et al.* (S136) presented the framework SPLEmma, which deals with feature-oriented SPL evolution. This framework follows a Model Driven Engineering approach to control the evolution. Thus, since it is a controlled SPL evolution, the SPL maintainer defines the authorized operations of evolution.

Around 54.2% of the approaches are manual (see Table 4.10), such as the approach revealed by Pena *et al.* (S79). This approach deals with describing, understanding, and analyzing evolving SPL without a tool support.

4.4.9 In which context the approach is applied? (RQ1.9)

The results for RQ1.9 shown that most of the approaches, around 93.6% are from academia (see Table 4.10). For instance, the approach from Schmid and Eichelberger (S92). They performed an academic study, which focus on difficulties that arise when migrating systems from development time variability to run-time variability.

Finally, the results revealed that around 8.4% of the approaches are from industry (see Table 4.10). An examples is the approach from Zhang *et al.* (S117).

The industrial study presented by Zhang *et al.* (S117) presents an approach to re-engineering, incrementally, an existing product family into an SPL following agile principles (refactoring and continuous integration). However, they claim that the company does not need to migrate towards agile software development.

The following Section presents the analysis of the results from the systematic mapping study, created based on the combination of the different sub-research questions.

4.4.10 Mapping Results

In order to analyze the results, we combine each one of the research questions related to *why*, *when*, *where*, *what*, and *how* SPL evolves with the research questions related to SPL life cycle, evaluation procedure, tool support, and context which the approach was applied.

Figure 4.2 shows the mapping results obtained from RQ1.1 (why SPL evolve) in comparison to RQ1.6 (SPL life cycle) and RQ1.7 (evaluation procedure). These results may indicate that:

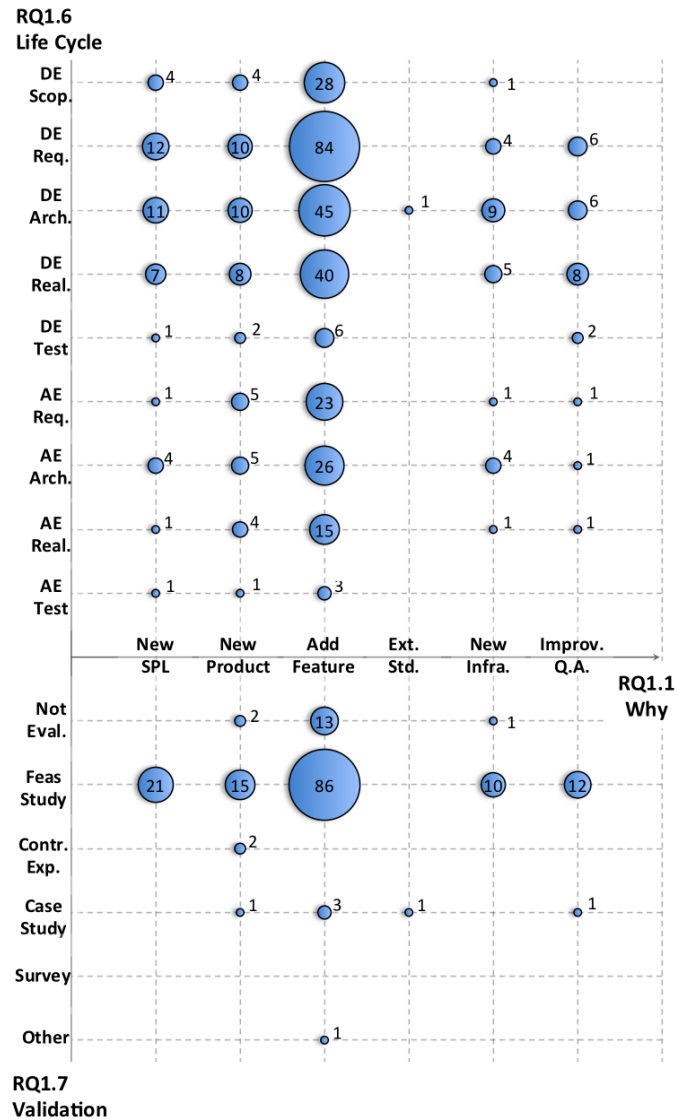


Figure 4.2: Bubble Chart for the combination of RQ1.1 (Why) by RQ1.6 (SPL Life Cycle) and RQ1.7 (Evaluation)

- *RQ1.1 (why) by RQ1.6 (SPL life cycle)*: Most of the approaches (59.15%) focus on add, remove or change features in the SPL domain requirements phase. Also, few approaches (11.26%) deal with evolution within test in domain and application engineering and few approaches (15.49%) deal with evolution within realization phase of the application

engineering. Moreover, few approaches (21.83%) deal with the evolution of requirements within the application engineering.

- *RQ1.1 (why) by RQ1.7 (evaluation)*: Most of the studies (60.56%) focus on add, remove or change features and were evaluated using a feasibility study. Also, few approaches were evaluated through case studies and controlled experiments (5.63%) and none approach was evaluated using survey.

Figure 4.3 shows the mapping results obtained from RQ1.1 (why SPL evolve) in comparison to RQ1.8 (tool support) and RQ1.9 (context). These results may indicate that:

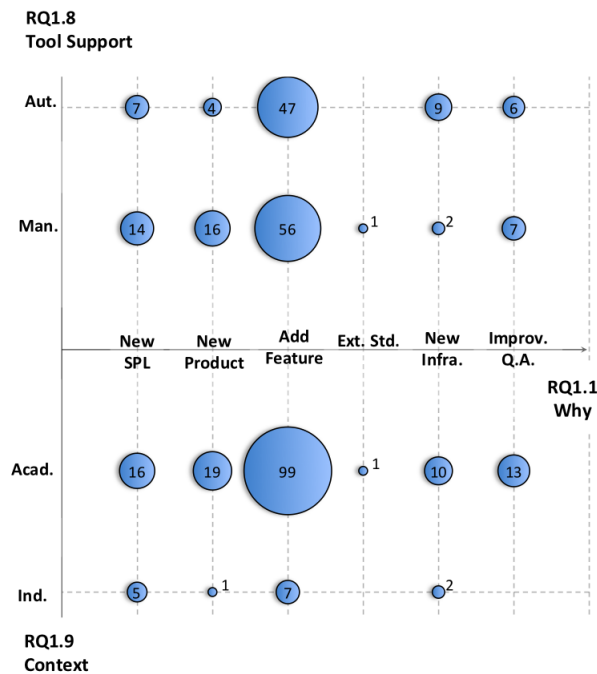


Figure 4.3: Bubble Chart for the combination of RQ1.1 (Why) by RQ1.8 (Tool) and RQ1.9 (Context)

- *RQ1.1 (why) by RQ1.8 (tool)*: Most of the presented approaches (39.43%) are manual and focus on add, remove or change features.
- *RQ1.1 (why) by RQ1.9 (context)*: Most of the studies (69.71%) were developed in the academia and focused on add, remove or change features.

The combination of RQ1.1 by RQ1.6, RQ1.7, RQ1.8, and RQ1.9 revealed that there is a lack of approaches dealing with extend standards, new version of infrastructure and improvements in quality attributes.

Figure 4.4 shows the mapping results obtained from RQ1.2 (when SPL perform the evolution) in comparison to RQ1.6 (SPL life cycle) and RQ1.7 (evaluation procedure). These results may indicate that:

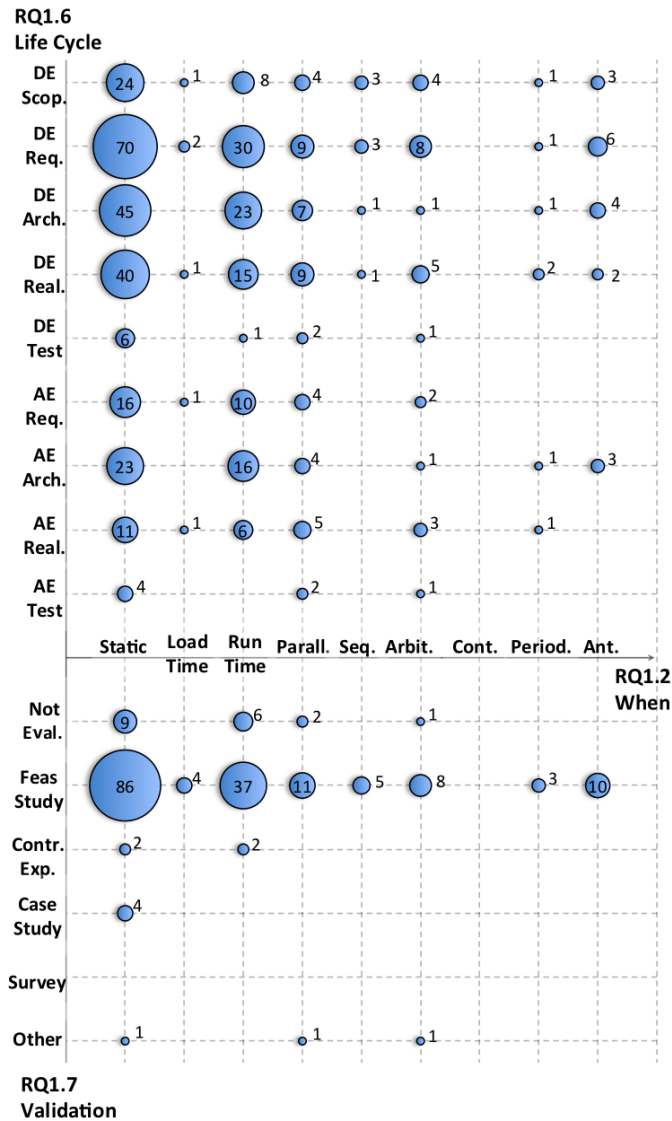


Figure 4.4: Bubble Chart for the combination of RQ1.2 (When) by RQ1.6 (SPL Life Cycle) and RQ1.7 (Evaluation)

- *RQ1.2 (when) by RQ1.6 (SPL life cycle)*: Most of the approaches (49.29%) focus on domain requirements phase with a static SPL evolution. Moreover, few studies (11.97%) deal with test for SPL Domain and Application engineering.

- *RQ1.2 (when) by RQ1.7 (evaluation)*: Most of the studies (60.56%) deal with SPL static evolution evaluated through a feasibility study. Also, few approaches (5.63%) were evaluated through case studies and controlled experiments and none approach was evaluated using survey.

Figure 4.5 shows the mapping results obtained from RQ1.2 (when SPL perform the evolution) in comparison to RQ1.8 (tool support) and RQ1.9 (context). These results may indicate that:

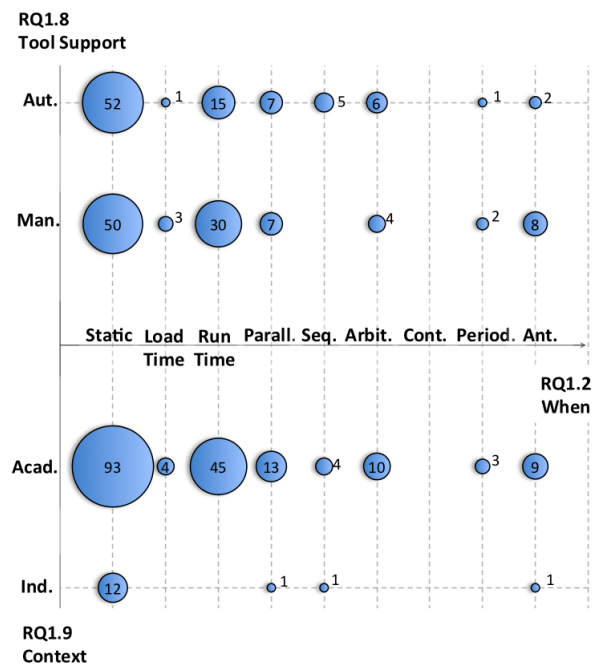


Figure 4.5: Bubble Chart for the combination of RQ1.2 (When) by RQ1.8 (Tool) and RQ1.9 (Context)

- *RQ1.2 (when) by RQ1.8 (tool)*: Most of the approaches (36.61%) are automated and focus on static SPL evolution. We could realize that the number of automatic approaches is slight greater than manual ones.
- *RQ1.2 (when) by RQ1.9 (context)*: Most of the approaches (65.49%) are on SPL static evolution and were applied within the academia. Also, there are more studies focusing on the academia than industry.

The combination of RQ1.2 by RQ1.6, RQ1.7, RQ1.8, and RQ1.9 shown that there is a lack of approaches for sequential, periodically and load time evolution in SPL. None approach deals with continuous evolution.

Figure 4.6 shows the mapping results obtained from RQ1.3 (where SPL perform the evolution) in comparison to RQ1.6 (SPL life cycle) and RQ1.7 (evaluation procedure). These results may indicate that:

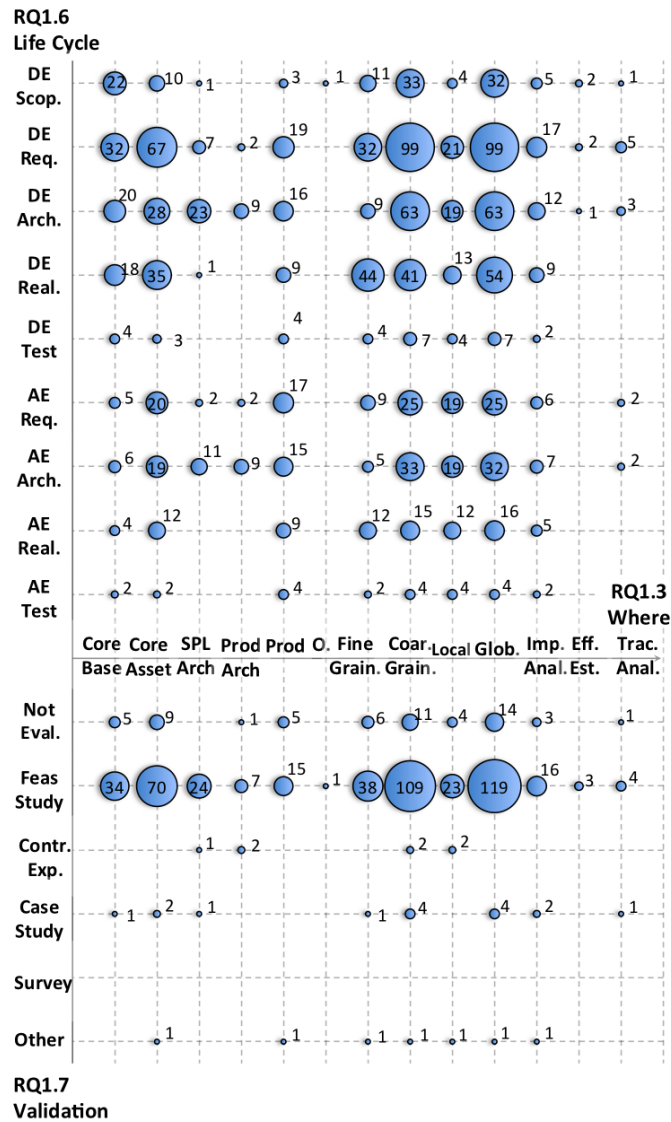


Figure 4.6: Bubble Chart for the combination of RQ1.3 (Where) by RQ1.6 (SPL Life Cycle) and RQ1.7 (Evaluation)

- RQ1.3 (where) by RQ1.6 (SPL life cycle)*: Most of the approaches (69.71%) have a global effect during the evolution and focus on coarse grained artifacts of domain requirements. Also, few studies (16.9%) support evolution of tests within the application engineering.

- *RQ1.3 (where) by RQ1.7 (evaluation)*: Most of the approaches (83.8%) have a global effect during the evolution and were evaluated through feasibility studies. Few approaches (16.19%) were evaluated through case studies and controlled experiments and none approach was evaluated using survey.

Figure 4.7 shows the mapping results obtained from RQ1.3 (where SPL perform the evolution) in comparison to RQ1.8 (tool support) and RQ1.9 (context). These results indicate that:

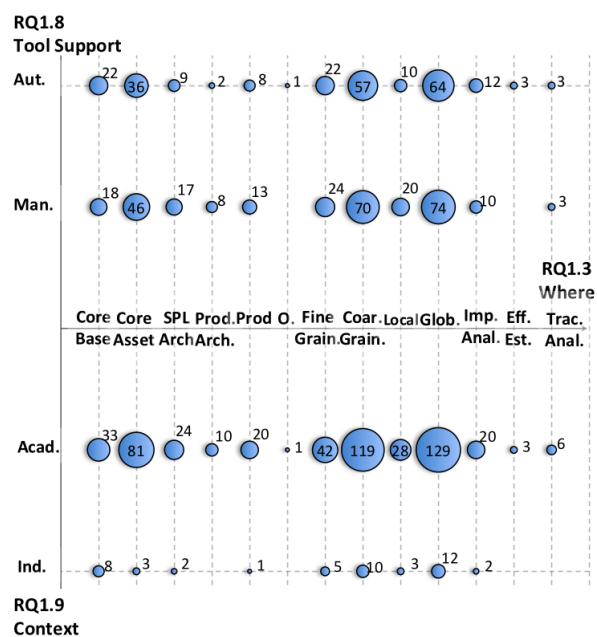


Figure 4.7: Bubble Chart for the combination of RQ1.3 (Where) by RQ1.8 (Tool) and RQ1.9 (Context)

- *RQ1.3 (where) by RQ1.8 (tool)*: Most of the approaches (52.11%) are manual and they have a global effect on the SPL during the evolution.
- *RQ1.3 (where) by RQ1.9 (context)*: Most of the approaches (90.84%) were performed in the academia and they deal with the global impact on SPL when it is evolving.

In general, the combination of RQ1.3 by RQ1.6, RQ1.7, RQ1.8, and RQ1.9 presented a lack of approaches dealing with traceability analysis, effort estimation and evolution in product architecture.

Figure 4.8 shows the mapping results obtained from RQ1.4 (what type of evolution) in comparison to RQ1.6 (SPL life cycle) and RQ1.7 (evaluation procedure). These results may indicate that:

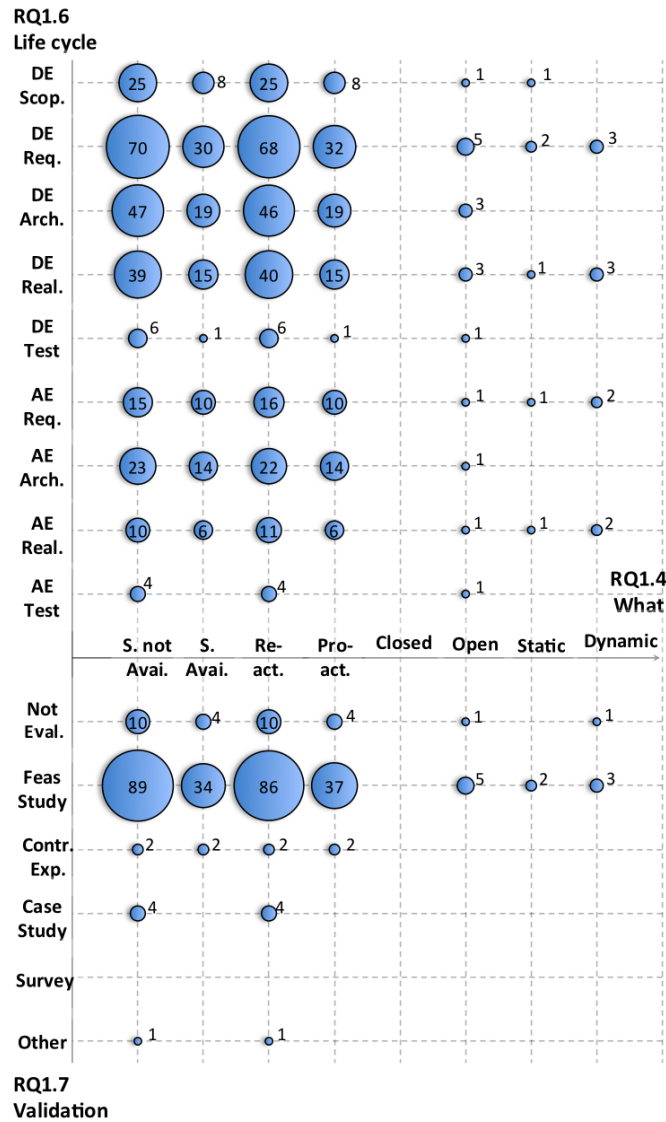


Figure 4.8: Bubble Chart for the combination of RQ1.4 (What) by RQ1.6 (SPL Life Cycle) and RQ1.7 (Evaluation)

- *RQ1.4 (what) by RQ1.6 (SPL life cycle)*: Most of the approaches (49.29%) deal with domain requirements engineering evolution in systems that are not always available during the evolution process. Few studies (16.9%) deal with evolution of tests for SPL domain and application engineering.

- *RQ1.4 (what) by RQ1.7 (evaluation)*: Most of the approaches (62.67%) deal with evolution of systems that are not always available during the evolution process and they are evaluated using feasibility studies. Moreover, there is a lack of approaches (11.26%) evaluated through case studies and controlled experiments and none approach was evaluated using survey.

Figure 4.9 shows the mapping results obtained from RQ1.4 (what type of evolution) in comparison to RQ1.8 (tool support) and RQ1.9 (context). These results may indicate that:

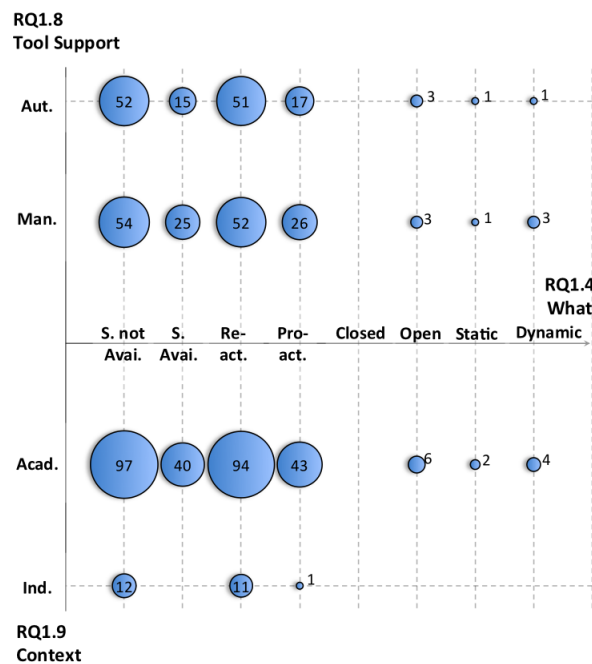


Figure 4.9: Bubble Chart for the combination of RQ1.4 (What) by RQ1.8 (Tool) and RQ1.9 (Context)

- *RQ1.4 (what) by RQ1.8 (tool)*: Most of the approaches (38.02%) are manual and they deal with systems that are not always available during the evolution process.
- *RQ1.4 (what) by RQ1.9 (context)*: Most of the approaches (68.3%) are from academia and they deal with evolution of systems that are not always available during the evolution process.

We could realize by combining RQ1.4 by RQ1.6, RQ1.7, RQ1.8, and RQ1.9 that few studies support safe dynamic and static evolution and also few approaches have an open framework. Moreover, none approach deals with the evolution of closed frameworks.

Figure 4.10 shows the mapping results obtained from RQ1.5 (how SPL approaches support the evolution) in comparison to RQ1.6 (SPL life cycle) and RQ1.7 (evaluation procedure). These results may indicate that:

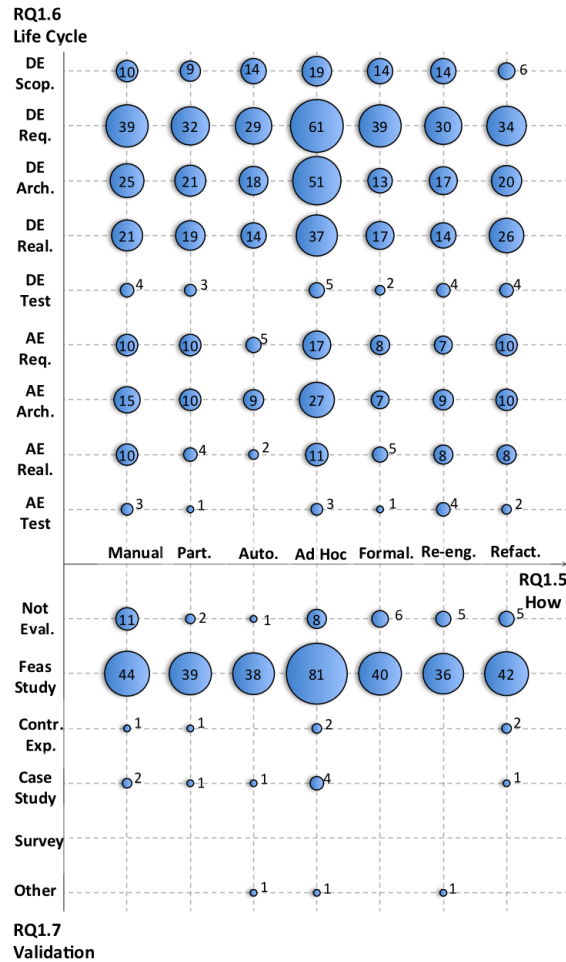


Figure 4.10: Bubble Chart for the combination of RQ1.5 (How) by RQ1.6 (SPL Life Cycle) and RQ1.7 (Evaluation)

- *RQ1.5 (how) by RQ1.6 (SPL life cycle)*: Most of the approaches (42.95%) focus on the evolution of domain requirements engineering in an *ad-hoc* way. Few studies (22.94%) deal with evolution of tests for domain and application SPL engineering.
- *RQ1.5 (how) by RQ1.7 (evaluation)*: Most of the approaches (57.04%) are *ad-hoc* and they are evaluated through feasibility studies. There is a lack of approaches (10.56%) evaluated

through case studies and controlled experiments and none approach was evaluated using survey.

Figure 4.11 shows the mapping results obtained from RQ1.5 (how SPL approaches support the evolution) in comparison to RQ1.8 (tool support) and RQ1.9 (context). These results may indicate that:

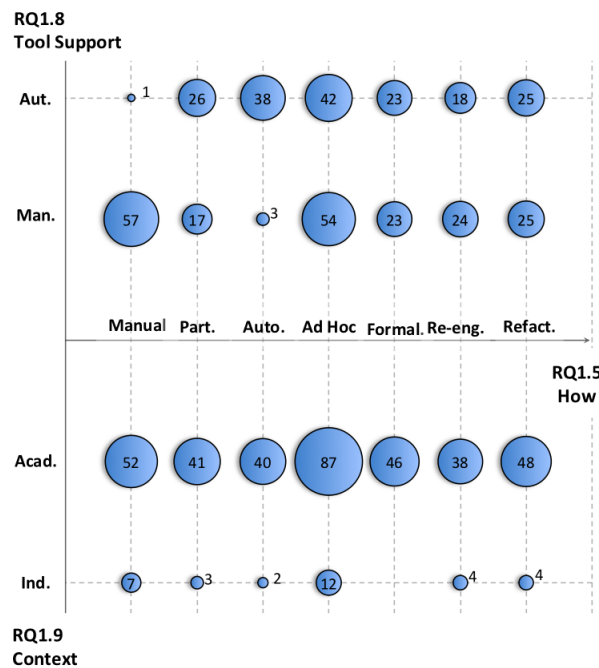


Figure 4.11: Bubble Chart for the combination of RQ1.5 (How) by RQ1.8 (Tool) and RQ1.9 (Context)

- *RQ1.5 (how) by RQ1.8 (tool)*: Most of the approaches (40.14%) are manual and do not have tool support.
- *RQ1.5 (how) by RQ1.9 (context)*: Most of the approaches (61.26%) are *ad-hoc* and from academia. There is a lack of industry studies.

The main findings, herein represented by $F(\text{number})$, from this mapping shown a lack of approaches that deal with:

- **F(1)**. evolution of requirements within SPL application engineering
- **F(2)**. evolution of the code within SPL application engineering

- **F(3)**. evolution of tests within the SPL domain and application engineering
- **F(4)**. evolution of standards support
- **F(5)**. effort estimation and traceability analysis (mainly for code of the SPL domain and application engineering)
- **F(6)**. evolution of closed frameworks
- **F(7)**. static and dynamic SPL safety evolution
- **F(8)**. evaluation through case studies
- **F(9)**. controlled experiments and surveys
- **F(10)**. approaches applied within the industry context

In addition, the findings also show that most of the approaches deal with:

- **F(11)**. evolution in an *ad-hoc* manner
- **F(12)**. coarse grained artifacts
- **F(13)**. global evolution (affecting the whole SPL)
- **F(14)**. evolution within the requirements phase (in an *ad-hoc* manner and through feasibility studies)
- **F(15)**. static evolution (evolution within systems that do not need to stay always available)
- **F(16)**. reactive evolution
- **F(17)**. evaluation through feasibility studies
- **F(18)**. manual (without tool support) evolution
- **F(19)**. academic context

We built Table 4.11 to summarize our findings by associating each *finding* with *why*, *when*, *where*, *what*, and *how* SPLs evolve.

4.4.11 Threats to Validity

It was identified some threats to the validity of our study. They are presented as follows: **I. Research questions**: The defined research question (and sub-research questions) might not cover the whole SPL evolution field. As we considered this as a feasible threat, we had several discussion meetings with project members from our research group³ and experts in the area in order to calibrate the questions. Thus, we attempted to address the most considered issues in the field, based on an previous defined taxonomy for software change (Buckley *et al.*, 2005) and based on possible evolution scenarios for SPL (Bosch, 2000); **II. Publication bias**: Within

³<http://rise.com.br>

Table 4.11: Association of the main findings with Why/When/Where/What/How SPLs Evolve.

Finding	Why	When	Where	What	How
F(1)	X				
F(2)	X				
F(3)	X	X	X	X	X
F(4)	X				
F(5)			X		
F(6)				X	
F(7)				X	
F(8)	X	X	X	X	X
F(9)	X	X	X	X	X
F(10)	X	X	X	X	X
F(11)					X
F(12)			X		
F(13)			X		
F(14)		X		X	X
F(15)		X		X	
F(16)				X	
F(17)	X	X	X	X	X
F(18)	X		X	X	X
F(19)	X	X	X	X	X

this systematic mapping study, we cannot assure that all papers addressing SPL evolution were selected. It is possible that we might have missed some relevant paper over the searching process. However, we mitigated this threat by performing a manual search on relevant journals and conferences in the area; **III. Period of the Selected Studies:** Before 1996, the community may have used the term *domain engineering* as a synonym to SPL. Moreover, since the last search for papers was performed in January of 2015, some relevant papers published after this date were not included in this systematic mapping study. To mitigate these threats, we presented a well-defined review protocol which allows to efficiently update and extend this systematic mapping study within papers published before 1996 and after 2015; **IV. Unfamiliarity with other fields:** Since the terms that were chosen in search strings might have synonyms, such as SPL, it is possible that we overlooked some papers, including papers from other areas.

4.5 Chapter Summary

As shown in this systematic mapping study, several approaches have been proposed to evolve SPL. However, so far, there is no systematic mapping study summarizing the whole existing

approaches that deal with SPL evolution.

This systematic mapping study summarized the existing information regarding SPL evolution. From an initial set of 1,940 papers, a total of 142 research papers were selected for the systematic mapping study, and the results obtained allowed to extract some conclusions regarding the state-of-the-art in the field.

This systematic mapping study summarized the existing information regarding SPL evolution. We started with an initial set of 1,940 papers after the first search. Later, after the filtering process, a total of 142 research papers were selected for the systematic mapping study. The systematic mapping study results allowed us to understand the state-of-the-art within SPL evolution.

The consolidation of the main findings from both industrial empirical studies evaluating the applicability of LL within SPL and the the systematic mapping study are presented in Table 4.12.

The 1st finding (“*Most of the SPL Evolution is Target to the Domain Engineering Assets*”) was revealed and confirmed in all of the three studies. The first and second industrial empirical studies revealed that most of the evolution activities are focused on common and variable assets. The systematic mapping study revealed that most of the SPL evolution approaches deal with the evolution of domain coarse grain artifacts.

The 2nd finding (“*There is a Need to Keep Constant or Improve the Quality*”) was revealed by the first industrial empirical study, partially confirmed within the second industrial empirical study, and confirmed within the systematic mapping study. The first industrial empirical study confirmed through the KPSS test and regression analysis that there is a decrease of quality within the SPL assets. The second industrial empirical study partially did not confirmed this finding through the KPSS test, but it confirmed through the regression analysis. The systematic mapping study confirmed this finding by revealing that there is a lack of approaches dealing with tests for domain and application engineering. Tests can improve the SPL quality (Young and Pezze, 2005), thus, there is a need to improve the SPL quality during the evolution.

The 3rd finding (“*There is a Need to Systematize the SPL Evolution Process*”) was revealed by the systematic mapping study. It revealed that most of the SPL evolution approaches are *ad-hoc*. The first and second industrial empirical studies also confirmed this finding, since both companies did not systematize their SPL evolution processes.

The 4th finding (“*There is a Need to Systematize Evolution of Requirements*”) was also revealed by the systematic mapping study. The systematic mapping study revealed that most approaches deal with the SPL requirements evolution in an *ad-hoc* manner. The first and second industrial empirical studies also confirmed this finding, since both companies did not systematize their SPL requirements evolution processes.

The 5th finding (“*There is a Need to Improve Tests within SPL*”) was also revealed by the systematic mapping study. The systematic mapping study revealed that there is a lack of approaches dealing with the evolution of SPL tests. The first and second industrial empirical studies also confirmed this finding, since both companies did not evolve their tests within their SPLs.

The 6th finding (“*There is a need for a tool support*”) was revealed by the systematic mapping study. The systematic mapping study revealed that there is a lack of tool support for the SPL evolution approaches. The first and second industrial empirical studies also confirmed this finding, since both companies did not use any kind of tool support to evolve their SPL assets.

All the findings from Table 4.12 are important and need to be treated to improve the SPL evolution process. In this Thesis, we decided to cope with the findings numbers 3 and 4, which led to the systematization of the SPL evolution processes, more specific, the SPL requirements. By systematizing the SPL evolution process we believe that we will also cope, indirectly, with the SPL quality (finding number 2).

Table 4.12: Findings Consolidation

Main Findings	1 st Industrial Empirical Study	2 nd Industrial Empirical Study	Systematic Mapping Study
1. Most of the SPL Evolution is Target to the Domain Engineering Assets	Source / Confirmed	Source / Confirmed	Source / Confirmed
2. There is a Need to Keep Constant or Improve the Quality of the SPL	Source	Partially Confirmed	Confirmed
3. There is a Need to Systematize the SPL Evolution Process	Confirmed	Confirmed	Source
4. There is a Need to Systematize Evolution of Requirements	Confirmed	Confirmed	Source
5. There is a Need to Improve Tests within SPL	Confirmed	Confirmed	Source
6. There is a need for a tool support	Confirmed	Confirmed	Source

In order to cope with the mentioned gap and deal with the SPL RE evolution in an systematic way, it was proposed guidelines for specifying and evolving the SPL requirements as shown in the next Part of this Thesis.

PART IV

**Guiding Software Product Lines Evolution
based on Requirements Engineering
Activities**

5

Feature-Driven Requirements Engineering (FeDRE) Approach

Defining requirements to determine what is to be developed is generally accepted as a vital but difficult part of software development. Establishing the driving architectural requirements not only simplifies the design and implementation phases but also reduces the number of errors detected in later stages of the development process, thus reducing the risk, duration and budget of the project (Jones, 1991).

The specification of requirements in SPL (Clements and Northrop, 2002) development faces some challenges, since it is necessary to deal with common, variable, and product-specific requirements, not only for a single product but also for the whole set of products in the family. Thus, to deal with these challenges, this Chapter presents the Feature-Driven Requirements Engineering (FeDRE) approach for specifying SPL requirements. This approach helps the requirements engineer to specify, in a systematic way, the SPL requirements driven by the feature model.

The remainder of this Chapter is organized as follows: Section 5.1 introduces the main challenges in the SPL requirements specification. Section 5.2 discusses related work. Section 5.3 presents our feature-driven requirements engineering approach. Section 5.4 evaluates the feasibility of the approach through an empirical study conducted to develop an SPL of mobile applications for emergency notifications. Finally, Section 5.5 presents the Chapter summary.

5.1 Introduction

One fundamental aspect of engineering SPLs is to apply RE practices to deal with the scoping and specification of the SPL in both the Domain Engineering and Application Engineering

processes.

In the Domain Engineering process, the RE activities are intended to define the extent of the SPL in order to determine its products (scoping), and also to identify common, variable, and product-specific features throughout the SPL. The specification of the requirements needed to deploy features must also be specified in a systematic manner by establishing explicit traceability between features and requirements. In the Application Engineering process, the RE activities are intended to specify the requirements for a particular product in the product family. It is therefore important to determine which requirements from the SPL are relevant to the product to be developed (common and variant feature selection), and also to refine or to add new specific requirements, not present in the SPL (delta requirements).

Most of the approaches that deal with RE in SPL development tend to include variability information in traditional requirements models (*e.g.*, use case diagrams) (Moon *et al.*, 2005) or to extract feature models (Kang *et al.*, 1990) from requirements specifications by following a bottom-up strategy (Asadi *et al.*, 2011; Mussbacher *et al.*, 2012), which may increase the complexity within this activity. Some limitations of these approaches arise from the possibility of a large number of requirements and features making the specification of requirements hard to understand, maintain and prone to inconsistencies. The contribution of the proposed approach is that it circumscribes the requirements specifications in order to deal with complexity in a more effective way. Effectiveness is achieved by chunking the requirements activity based on areas of the feature model. It constrains the extent of the requirements specification at any one time to a more specific area of the SPL. The feature model is used as a basis principally because in the SPL community, features are first-class citizens, which are easily identifiable, well-understood, and more clear for SPL developers and domain experts to communicate. There is thus a strong need to define traceability links between these features and requirements, and whenever possible, to maintain the model and specification synchronized and consistent (Anquetil *et al.*, 2010; Heidenreich *et al.*, 2010; Alférez *et al.*, 2011).

In order to improve the SPL requirements phase, which was one of the gaps pointed from the previous studies (Alves *et al.* (2010) and Oliveira *et al.* (2015d)), the Feature-Driven Requirements Engineering (FeDRE) approach (Oliveira *et al.*, 2013, 2014) was defined and evaluated to aid developers in the Requirements Engineering (RE) activity for SPL development. This approach focuses on the specification of requirements at early stages, taking as input the scoping artifacts. Thus, the approach proposes a set of artifacts, activities, roles, and guidelines on the basis of the features to be developed. The focus is the requirements specification in the Domain Engineering activity. We further focus our description of FeDRE by starting once a feature model has been defined (in the scoping activity). The feature model is used as the main

artifact for the specification of the SPL requirements. However, the approach does not deal with Quality Attributes (QAs) (Montagud *et al.*, 2012) in the feature model. FeDRE has an activity to systematize the realization of features in terms of use cases. This activity specifies requirements but also establishes traceability between the features and the requirements. This allows us to provide variability mechanisms at the requirements level (by using use cases and alternative scenarios) according to the chunk of the feature model that these requirements specify. The main contributions of FeDRE is an RE approach that: 1) systematically realizes features into requirements by considering the variability captured in the feature model; and 2) breaks the top-down driven paradigm through use of the feature model in order to prioritize features according to significant areas of the SPL. A first evaluation of FeDRE was performed through an empirical study within an SPL project, where the perceived ease of use, perceived usefulness, effectiveness and efficiency of the approach were evaluated.

5.2 Related Work

Several models and techniques that deal with the specification of requirements in SPL development have been proposed over the last few years. Some of these proposals were analyzed by using a comparison criteria in order to discover how the current approaches cover the RE activity to model SPL requirements. The comparison criteria were formed of four main criteria. The first analysis the *SPL activities* supported in the development (Domain Engineering, and Application Engineering). The second criterion encompasses the *RE activities* that were used in the RE approaches according to the disciplines (elicitation, specification, analysis, verification and management) that guide an RE process (Clements and Northrop, 2002). In the third criterion, we analyzed which *artifacts* were employed to model the requirements. Finally, it was analyzed *how the process was defined*. The analysis of “how the process was defined” was performed analyzing three sub-criteria: whether the approach provides guidelines, whether the approach defines roles, and whether the approach has well defined inputs and outputs. These three sub-criteria were selected since they help in the systematization of the process.

It was analyzed several RE approaches for SPL development, and we found a distinct set of approaches and techniques (Table 5.1). Summarizing, in many cases, the scoping and requirements specification activities are considered as independent activities. According to John and Eisenbarth (2009), well-defined relationships and interfaces between scoping and requirements artifacts should be defined in order to reduce rework. To mitigate this problem, FeDRE considers the scoping artifacts as the starting point and defines guidelines to conduct the

SPL requirements specification driven by the scoping artifacts. Another important aspect is the strategy followed to specify the requirements. Several approaches, such as use cases (*i.e.*, (Griss *et al.*, 1998; Eriksson *et al.*, 2005)) or goal models adapted to the SPL domain (*i.e.*, (Asadi *et al.*, 2011; Mussbacher *et al.*, 2012)), extend RE models and extract feature models from these RE models. In our view, SPL developers and domain experts are more familiar with the concept of features and variability modeling.

Table 5.1: Comparative among current RE proposals from SPL.

Approach	SPL Processes	RE Disciplines	Artifacts	Process Definition
FeatuRSEB (Griss <i>et al.</i> , 1998)	Domain engineering	Elicitation, modeling, analysis	Use case model, feature model	Partially (guidelines, inputs and outputs)
PLUSS (Eriksson <i>et al.</i> , 2005)	Domain engineering, application engineering	Elicitation, modeling, analysis, management	Feature model, use case, change case	Partially (guidelines, inputs and outputs)
VML4RE (Alf�rez <i>et al.</i> , 2011)	Domain engineering, application engineering	Elicitation modeling, analysis, management	Feature model, use cases, activity diagrams	Partially (inputs and outputs)
MSVCM (Bonif�cio and Borba, 2009)	Domain engineering, Application engineering	Modeling, analysis, management	Use case model, feature model, product configuration, configuration knowledge	Partially (inputs and outputs)

continued on next page

continued from previous page

Approach	SPL Processes	RE Disciplines	Artifacts	Process Definition
Pulse-CDA (Bayer <i>et al.</i> , 1999a)	Domain engineering	Elicitation, modeling, analysis	Domain analysis model, use cases	Partially (inputs and outputs)
DREAM (Moon <i>et al.</i> , 2005)	Domain engineering	Elicitation, modeling, analysis	PR-Context matrix, use cases	Partially (guidelines, inputs and outputs)
AoURN (Mussbacher <i>et al.</i> , 2012)	Domain engineering, application engineering	Elicitation, modeling, analysis	Stakeholder goal model, feature model, feature impact model, feature scenario model	Partially (inputs and outputs)
FORML (Shaker <i>et al.</i> , 2012)	Domain engineering	Modeling	Feature model, behavior model	Partially (inputs and outputs)
FEDRE (Oliveira <i>et al.</i> , 2013) (Oliveira <i>et al.</i> , 2014)	Domain engineering	Elicitation, modeling, management	Feature model, feature specification, product map, glossary, traceability matrix, use cases	Complete (guidelines, inputs and outputs)

5.3 Feature-Driven Requirements Engineering Approach For SPL

The Feature-Driven Requirements Engineering (FeDRE) approach for SPL has been defined by considering the feature model as the main artifact for specifying SPL requirements. The aim of the approach is to perform the requirements specification by systematically utilizing the features identified in the SPL domain through the use of guidelines that establish traceability links between features and requirements. By domain, we mean the context in which the family

of products or functional areas across the products exhibits common, variable or specific functionalities.

The main activities of the FeDRE approach are: Scoping and Requirements Specification for Domain Engineering. The Requirements Specification for Application Engineering is not yet handled by the approach. Figure 5.1 shows the first two activities in FeDRE, which are detailed in this Chapter. The following roles are involved in these activities: Domain Analyst, Domain Expert, Market Expert and the Domain Requirements Analyst. FeDRE uses the *Software and Systems Process Engineering Meta-model (SPEM)*¹ for presenting its overview and guidelines.

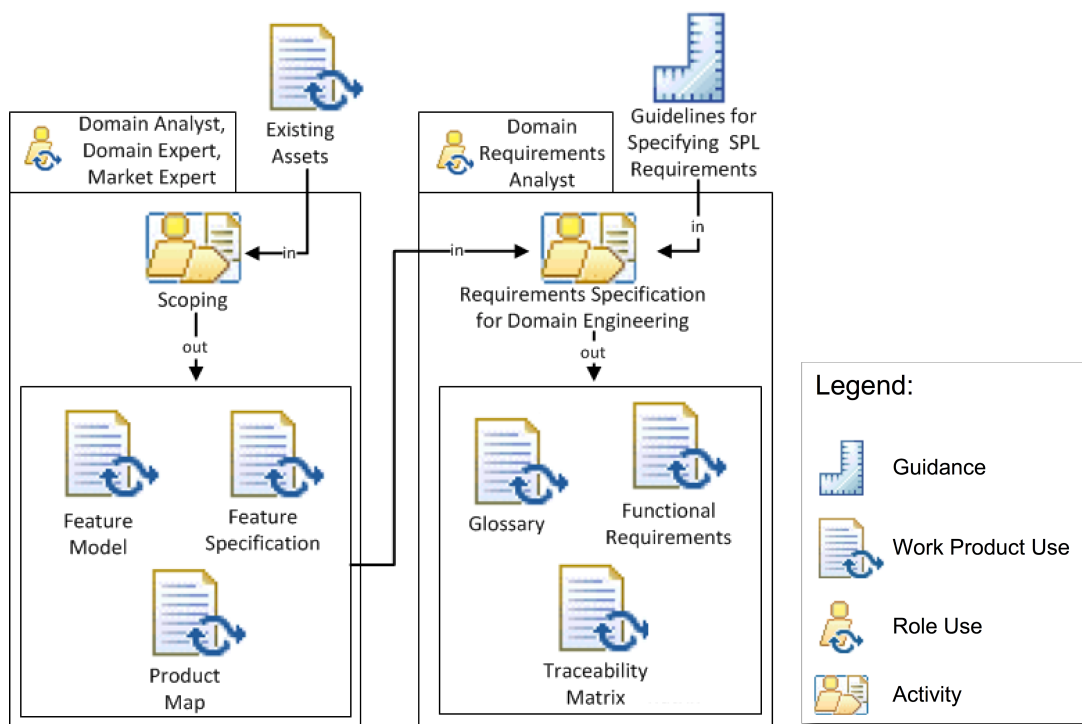


Figure 5.1: Overview of the FeDRE approach.

5.3.1 Scoping

The first activity performed in FeDRE is the Scoping. This determines not only what products to include in an SPL but also whether or not an organization should launch the SPL. According to Bosch (2000), the Scoping activity consists of three levels: product portfolio Scoping, domain Scoping, and asset Scoping. Product portfolio Scoping determines which products and features

¹<http://www.omg.org/spec/SPEM/2.0/>
 SPEM allows modeling an approach based on predefined profiles.

5.3. FEATURE-DRIVEN REQUIREMENTS ENGINEERING APPROACH FOR SPL

should be included in an SPL. Domain Scoping defines the functional areas and subareas of the SPL domain, while Asset Scoping identifies assets with costs and benefits estimated for them.

In FeDRE, the Domain Expert and the Market Expert perform the product portfolio Scoping. The Domain Expert and Domain Analyst perform the Domain Scoping. Finally, all the roles in the Scoping activity perform the Asset Scoping.

Three main artifacts are produced as a result of the Scoping activity: the Feature Model, the Feature Specification, and the Product Map, using the Existing Assets (if any) as the input artifact. These three artifacts will drive the SPL requirements specification for domain engineering. Details of the Scoping activity are shown in Figure 5.2. Each of these artifacts (input and outputs) is detailed below.

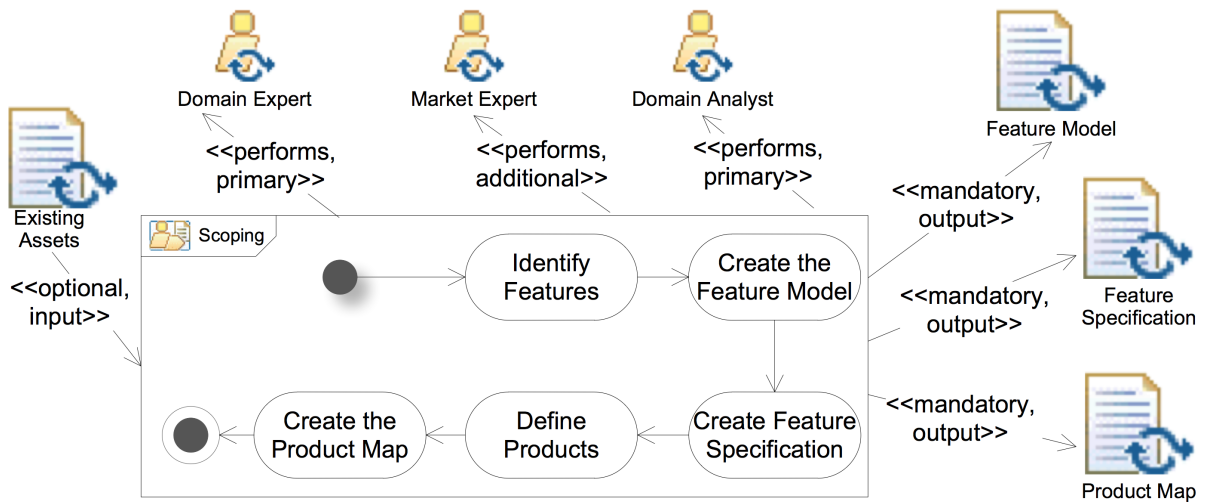


Figure 5.2: Detailed Scoping Activity.

5.3.1.1 Existing Assets

FeDRE supports the extractive, reactive, and proactive SPL adoption approaches. When performing an extractive or reactive SPL adoption (Krueger, 2002b), existing assets (e.g., user manual or existing systems) help the Domain Analyst and the Domain Expert to identify the features and products in the SPL. Otherwise, a proactive approach can be followed to build the SPL from scratch.

5.3.1.2 Feature Model

Feature modeling is a technique that is used to model common and variable properties, and can be used to capture, organize and visualize features in the SPL. The Domain Analyst and the

Domain Expert identify features using existing assets as input or by eliciting information from experts and from the Market Expert. A Feature Model diagram (Kang *et al.*, 1990) identifies features, SPL variations, and constraints among the features in the SPL.

5.3.1.3 Feature Specification

The Domain Analyst is responsible for specifying the features using a feature specification template. This template captures the detailed information of the features and maintains traceability with all the artifacts involved. According to the template, each feature must have a unique identifier *Feat id* and a *Name*. The values for the *Variability* field can be Mandatory, Optional, or Alternative, according to the feature specified (Table 5.2). The Priority of the feature should be High, Medium or Low. If the feature requires or excludes another feature(s), the *Feat id*(s) from the required or excluded feature(s) must be specified. If the feature has a *Parent* feature, the *Feat id* from the parent feature must be specified. The *Binding Time* can be compile time or runtime, according to the time that the feature will be included in a concrete product (Czarnecki and Eisenecker, 2000). The *Feature Type* can be concrete or abstract, and the *Description* is a brief explanation of the feature. Each one of these fields is necessary to represent a feature in FeDRE and to capture, mainly, the feature variability.

Table 5.2: Features Variability.

●	Mandatory Feature
○	Optional Feature
▲	Alternative Feature (OR) (one or more feature(s) can be selected)
⋈	Alternative Feature (XOR) (only one feature can be selected)

5.3.1.4 Product Map

Each of the identified features is assigned to the corresponding products in the SPL. The set of relationships among features and products produces the Product Map artifact, which describes all the features that are required to build a specific product in the SPL. It is usually represented as a matrix in which columns represent the products and rows represent the features. The Market Analyst, the Domain Analyst and the Domain Expert produce this artifact. An example of a product map is shown in Figure 5.3.

All these artifacts are the input for the Requirements Specification for Domain Engineering activity, which is described next.

Features / Products		SAVi Lite	SAVi Standard	SAVi Social	SAVi Pro	SAVi Ultimate
Access_Control				X	X	X
	Web_Access_Control			X	X	X
	Mobile_Access_Control			X	X	X
Contact		X	X	X	X	X
	Add_Contact	X	X	X	X	X
	Import_Contact		X	X	X	X
	Facebook_Import			X	X	X
	Twitter_Import			X	X	X
	Phone_Import		X	X	X	X
Destination		X	X	X	X	X
	SMS_Destination	X	X	X	X	X
	Twitter_Destination			X	X	X
	Facebook_Destination			X	X	X
	Email_Destination			X	X	X

Figure 5.3: Product Map Example.

5.3.2 Requirements Specification for Domain Engineering

This activity specifies the SPL requirements for domain engineering. These requirements allow realization of the features and desired products identified in the Scoping activity. The steps required to perform this activity are described in the Guidelines for Specifying SPL Requirements, Sub-Section 5.3.3 below.

The FeDRE approach was defined using (types of variabilities) and extending (through guidelines) the PLUS approach (Eriksson *et al.*, 2005), which represents requirements specifications as use case scenarios. The use case scenarios “*force requirements analysts to always think about interfaces since separate fields exist for describing actor and system actions*”. The approach supports the relationship between features and use cases. The feature variability is expressed within the use cases. FeDRE uses two types of variability from PLUS: i) use case variability, considering the whole use case as a variant; and ii) scenario variability, in which the variants are alternative scenarios of a use case. Within FeDRE these two types of variability are sufficient to capture the variations within SPL requirements. We have experienced in general that in SPLs the variability does not go beyond use case variability and scenario variability. The empirical study was also performed to evaluate this fact. It was also analyzed the Software Product Line Conference (SPLC), and in the majority of the approaches, the variability of the examples and industry projects could be solved with these two levels of requirements variability. So far, FeDRE is responsible for specifying requirements with a high level of abstraction, nevertheless, in the future, if we need more expressive variability mechanisms (*i.e.*, fine-grained variability)

we will consider to incorporate them. Moreover, FeDRE propose full guidelines for specifying the SPL requirements based on the features from the feature model. PLUS does not propose full guidelines for the SPL requirements specification.

When a requirement is identified or refined, it is necessary to determine whether it is a shared requirement for different products in the SPL, or whether it is a specific requirement of a single product. Shared requirements must also be classified into common and variable requirements. Common requirements are used throughout the SPL and variable requirements must be configured or parameterized in the specification of different variants of the SPL. In addition, some requirements may require or exclude other requirements, or may restrict possible configurations of other requirements. Feature models may help in handling the different types of dependencies among requirements, which can be complex and must be properly addressed.

The Requirements Specification for Domain Engineering activity is usually performed in an iterative and incremental manner. Sets of selected features from the Feature Model can therefore be defined as units of increments for the specification (different criteria may be used to choose features in a unit of increment, *e.g.*, priority of implementation, cost, QAs). This activity (Figure 5.4) uses the Feature Model, Feature Specification and Product Map as input artifacts and produces the Glossary, Functional Requirements and Traceability Matrix as output artifacts. Each of these output artifacts is detailed below.

5.3.2.1 Glossary

One important characteristic of an SPL is the presence of multiple stakeholders, domain experts, and developers. Thus, it is therefore necessary to have a common vocabulary for describing the relevant concepts of the domain. The Glossary describes and explains the main terms in the domain in order to provide the stakeholders with a common vocabulary and avoid misconceptions. It is represented as a two-column table containing the term to be defined and its description (see Table 5.4 in Section 5.4).

5.3.2.2 Functional Requirements

This artifact contains all the functional requirements identified (common or variable), for the family of products that constitute the SPL. Use cases are used to specify the SPL functional requirements (each functional requirement is represented as a use case), and the variations required can be related to the use case as a whole or to alternative scenarios inside a use case. FeDRE adapts the template used in (Eriksson *et al.*, 2005) in order to specify functional requirements as use cases, thus supporting both types of variability. The specification of

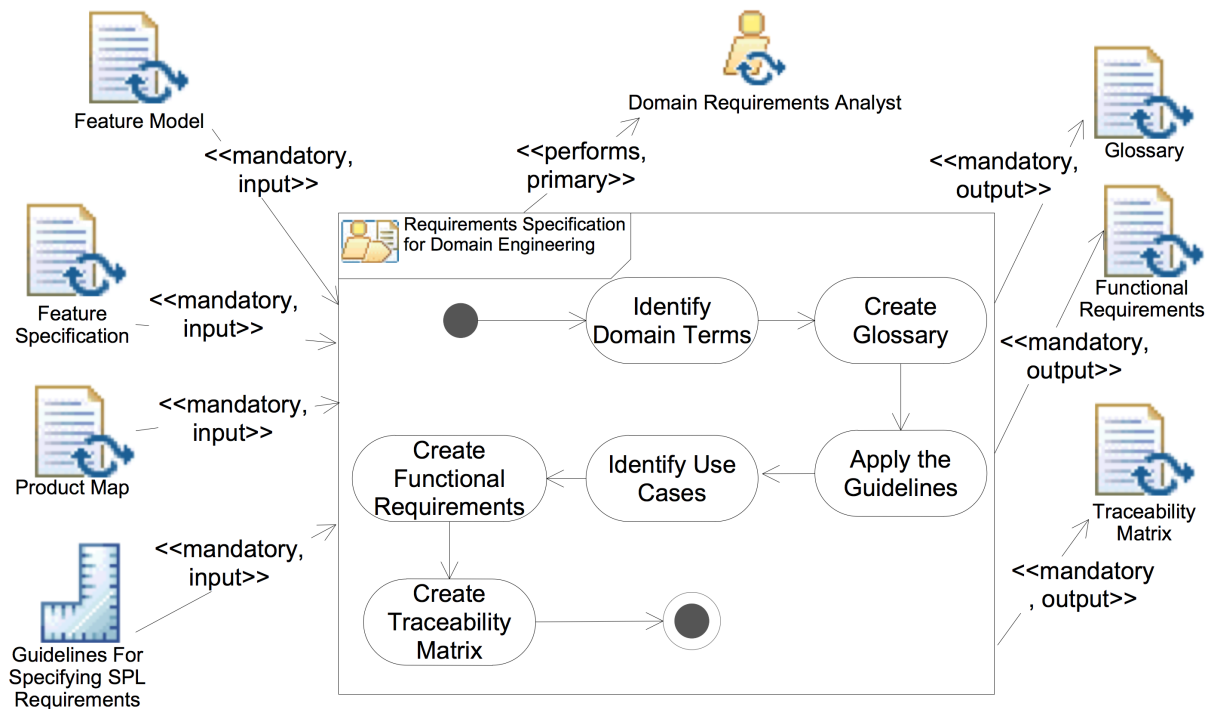


Figure 5.4: Detailed Requirements Specification Activity.

functional requirements follows the functional requirements template shown in Table 5.6 (Section 5.4). Each functional requirement has a unique *Use case id*, a *Name*, a *Description*, *Associated Feature(s)*, *Pre and Post-Conditions*, and the *Main Success Scenario*. A functional requirement can also be related to an Actor and may have Include and/or Extend relationships with other use case(s). Extends relationships should describe a condition for the extension.

5.3.2.3 Traceability Matrix

The Traceability Matrix is a matrix that contains the links among features and the functional requirements. The rows in the matrix show the features and the columns show the functional requirements, as shown in Table 5.5 (Section 5.4). This matrix is also useful as regards helping in the evolution of the requirements since each change in the feature model will be traced up to the requirements through the traceability matrix (and vice versa).

5.3.3 Guidelines for Specifying SPL Functional Requirements

The purpose of the guidelines is to guide the Requirements Analyst in the specification of SPL functional requirements for domain engineering. The guidelines are based on a meta-model (see

Figure 5.5) that represents the concepts involved when specifying use cases with alternative scenarios and the relationships among them.

The meta-model is used to maintain the traceability among all the elements and to facilitate understanding. The meta-model comprises the following elements:

- *RequirementsSpecification*: Is the container of all the elements in the specification
- *Feature*: This represents a feature from a variability model. Although it is not defined in this model, it is related to zero or many requirements
- *Requirement*: It is an abstract metaclass used to represent functional requirements
- *UseCase*: Represents a functional requirement. A *UseCase* is associated with a *Feature*, other *UseCases* through the *include*, *extend* or *inheritance* relationships, or with *Actors*. It contains a *Main Scenario* and zero or many *Alternative Scenarios*
- *UseCasePackage*: This is the container for a *UseCaseDiagram*
- *UseCaseDiagram*: This is a view for *Actors*, *UseCases* and *Relationships*
- *Actor*: Is an actor and can be related to other *Actors* or associated with *UseCases*
- *Relationship*: Represents the different types of relationships among *UseCases*, which are *Include*, *Extend* and *Inheritance*
- *Scenario*: This is an abstract metaclass used to represent the two types of scenarios for the *UseCase*, which are *MainScenario* and *AlternativeScenario*
- *MainScenario*: Represents the “normal flow” of steps for a *UseCase*
- *AlternativeScenario*: Represents an alternative set of steps for a *UseCase*. It can be associated with a *Feature* to represent the variability in the scenario
- *Step*: Represents a step in the *MainScenario* or *AlternativeScenario*.

The guidelines have been structured to specify functional requirements by addressing the following questions: i) **Which** features or set of features will be grouped to be specified by use cases? ii) **What** are the specific use cases for the feature or set of features? iii) **Where** should the use cases be specified? (when there is a set of features in a hierarchy, do we specify the use cases for each individual feature or only for the parent features?) and iv) **How** is the use case specified in terms of steps?

The guidelines consider four types of feature variability that may be present in the feature model, as shown in Table 5.2. Activities, tasks and steps are used in the process of specifying requirements for SPL as is shown in Figure 5.6. The first activity, *Identify Use Cases*, uses the Feature model as an input and generates two artifacts as an output, *Traceability Matrix* and *Use Case Diagram*. The second activity, *Specify Use Cases*, uses the two outputs from the previous activity plus a *Use Case Template* to generate the *Use Case Specification*. Figure 5.7 shows the

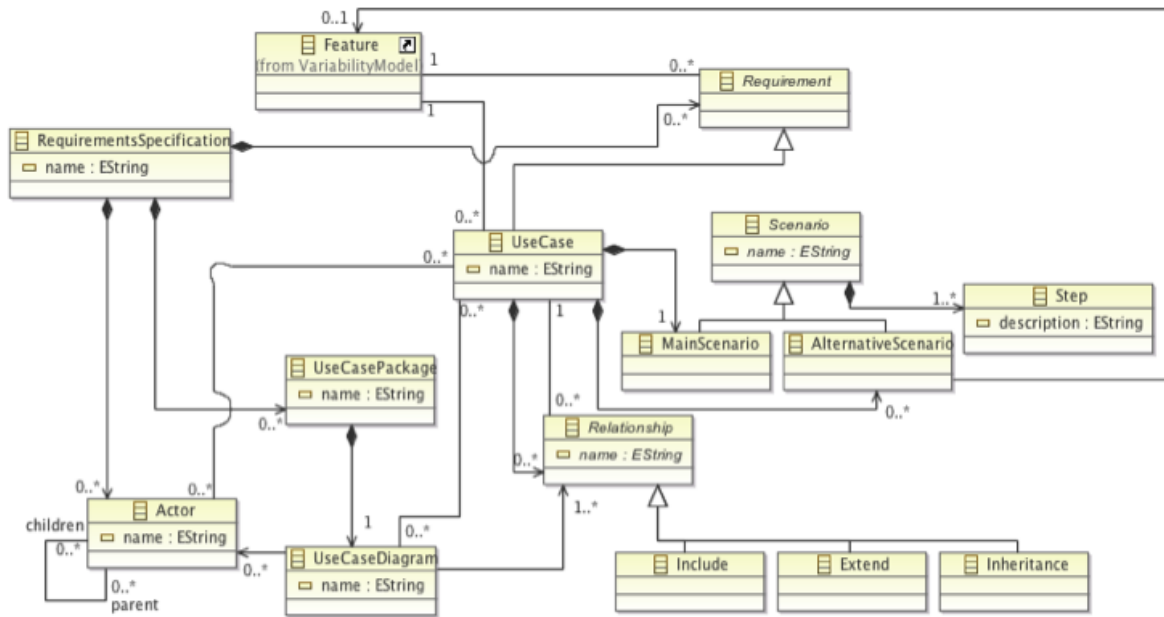


Figure 5.5: Meta-Model for SPL Requirements.

guidelines with the detailed steps of each task for specifying SPL functional requirements.

Next, we clarify some steps of the guidelines. According to the guidelines, *Root feature* is the first feature from a feature model hierarchy, *leaf feature* is the last feature from a feature model hierarchy, and *intermediate feature* is a non root feature neither a leaf feature. The Step 1.1 says that “*root mandatory features or intermediate mandatory features **must** have use cases*”. They must have use cases to handle the variability of children features. This variability is handle through their use cases alternative scenarios. The Step 1.2 says that “*Mandatory leaf features **may** have use cases, or can be specified as alternative scenarios from use cases in the parent feature*”. The guidelines leave this step as an optional step, where the requirements engineer can define use cases for these features, or the use cases will not exist and they will be specified as alternative scenarios from use cases identified within parent features. The Step 1.4 says that “*Use cases identified for leaf optional features **should be** specified as alternative scenarios from use cases in the parent feature*”. The guidelines also leave this step as an optional step, where the requirements engineer can define use cases for these features, or the use cases will not exist and they will be specified as alternative scenarios from use cases identified within parent features. The Step 3.1 says that “*Use cases identified for children features (that have the same parent) with similar behavior **must be** specified just once in the parent feature*”. This happens when a feature has children features that have similar behaviors. When these children features

share most of their steps from their use cases, they must be specified as alternative scenarios from use cases identified within the parent feature. The other steps from the guidelines are self explained in Figure 5.7.

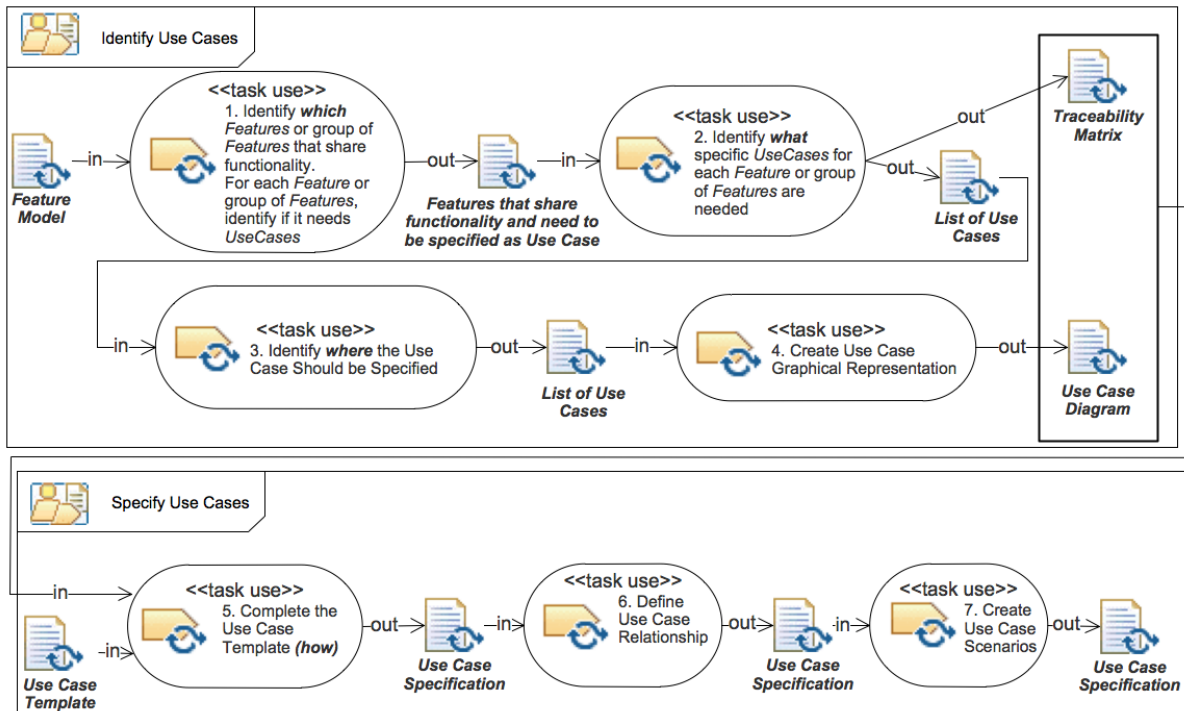


Figure 5.6: Overview of Activities, Tasks and Artifacts from the Guidelines.

5.4 Empirical Study

An exploratory empirical study to assess the usefulness of FeDRE was performed by following the guidelines presented in Wohlin *et al.* (2012). Besides this is a first evaluation of FeDRE for Domain Engineering, the obtained results make us to appreciate FeDRE as a promising approach. The stages of the empirical study are: design, preparation, collection of data, and analysis of data. Additionally it was included a subsection for the threats to validity.

5.4.1 Design of the empirical study

Firstly, the objectives and empirical study are planned. In order to define which objectives the empirical study would have, we applied the Goal-Question-Metric (GQM) (Basili *et al.*, 1994). After applying this approach, we stated that the goal of the empirical study was: to **analyze**

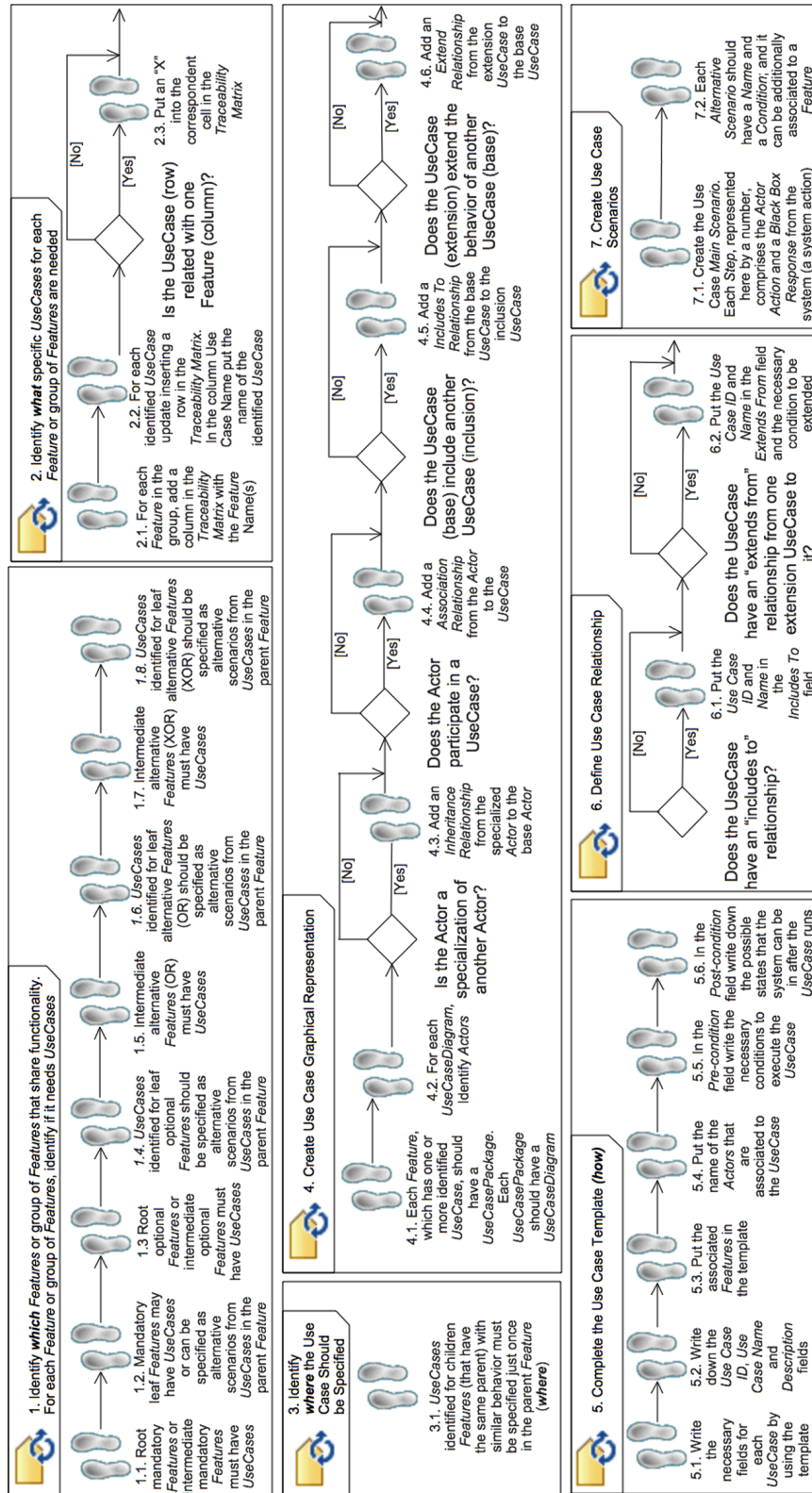


Figure 5.7: Guidelines For Specifying SPL Functional Requirements.

FeDRE **for the purpose** of evaluating it **with regard to** its perceived ease of use, and perceived usefulness **from the viewpoint** of a set requirements engineers.

The context of the empirical study is the requirements modeling of an SPL project. The SPL selected is for the mobile software called SAVi, which is an application that notifies and allows a mobile contact list to track a user in an emergency situation, sending a code by SMS and email to the contact list. We chose SAVi in order to apply FeDRE in an SPL project using an extractive / reactive SPL adoption.

There are **two subjective dependent variables**: *perceived ease of use* and *perceived usefulness*. To measure both variables after applying the FeDRE approach, it was used an existing measurement instrument proposed for the evaluation of requirements modeling methods based on user perceptions (Abrahão *et al.*, 2011). More specifically, the two perception-based variables were adapted from the aforementioned instrument, which were based on two constructs from the Technology Acceptance Model (TAM) (Davis, 1989):

- *Perceived Ease of Use (PEOU)*: the degree to which a person believes that using FeDRE will be effort-free. This variable represents a perceptual judgment of the effort required to learn and use the FeDRE approach;
- *Perceived Usefulness (PU)*: the degree to which a person believes that FeDRE will achieve its intended objectives. This variable represents a perceptual judgment of the FeDRE approach's effectiveness.

It was defined a set of items to measure these *perception-based variables*. These items were combined in a survey consisting of 7 statements. The items were formulated by using a 5-point Likert scale, using the opposing-statement question format. Various items within the same construct group were randomized to prevent systemic response bias. PEOU and PU were measured by using three and four items in the survey, respectively ².

We formulated the following **hypotheses**:

- $H1_0$: FeDRE is perceived as difficult to use, $H1_a$: FeDRE is perceived as easy to use.
- $H2_0$: FeDRE is perceived as not useful, $H2_a$ FeDRE is perceived as useful.

In addition, before the empirical study session, a requirements specification considered as the correct solution was defined. This requirements specification was created by three Ph.D. students. The aim of this agreed solution was to compare the subjects' solutions with the agreed

²The survey statements are available in Appendix C.1

solution in order to analyze the degree in which the subjects applied FeDRE in an effective and efficient way. **Four objective dependent** variables were defined with this aim in mind:

- *Effectiveness_UC*, which is calculated as the ratio between the number of right use cases that the subject identified and the total number of right use cases.
- *Effectiveness_SCEN*, which is calculated as the ratio between the number of right alternative scenarios that the subject identified and the total number of right alternative scenarios.
- *Efficiency_UC*, which is calculated as the ratio between the number of right use cases that the subject identified and the total time spent on the use cases identification.
- *Efficiency_SCEN*, which is calculated as the ratio between the number of right alternative scenarios that the subject identified and the total time spent on the alternative scenarios specification.

Table 5.3 shows the empirical study planning. Before the case study session, there was a 30 minutes **training session** to present an introduction of RE from SPL, and the use cases main concepts and notation (*i.e.*, use cases, actors, types of relations, etc.), the FeDRE method, and the objectives and procedures of the study. After the training session, the empirical study was performed. This session was composed of two tasks. When the subjects finished, they filled in a questionnaire about FeDRE.

Table 5.3: Planning.

	Group
Training (30 min)	Introduction to FeDRE
	Exercise explanation
Session (90 min)	Task 1: Identify Use Cases
	Task 2: Specify Use Cases
	FeDRE Questionnaire

Several **documents**³ were designed as instrumentation for the empirical study: slides for the training session, an explanation of the method, a data gathering form, and a questionnaire. These documents were used by the **subjects**, which were chosen for convenience from a group of software engineering research associates. The subjects were asked about their experience in the area, and the results showed that none of them had a previous background in this context. In consequence, it was not established a classification of subjects based on their experience in SPL

³The material is available at: <http://goo.gl/5wlmT0>

RE and it was not applied a leveraging questionnaire. The first session was composed of 8 Ph.D. students from the Universitat Politècnica de València, Spain, and the second was composed of 6 Ph.D. students from the Federal University of Bahia, Brazil.

5.4.2 Preparation of the empirical study

With regard to the Scoping activity, all the artifacts (*i.e.*, Feature Model, Feature Specification and Product Map) were created by one domain analyst and one domain expert, who were also assisted by a scoping expert with more than 6 years of experience in SPL scoping activities. The marketing analysis was carried out on the basis of other products, with a similar purpose to that of SAVi, which are available at the *AppStore*⁴. The functionalities of these products were included in the SAVi feature model, in which 27 features were identified.

Since the FeDRE approach is flexible to support the incremental requirements specification, a set of features was selected for the empirical study. The selection of these features was made on the basis of which features are present in most of the products in the Product Map and are easier to be implemented. Figure 5.8 shows an excerpt of the Feature Model and the features selected for the empirical study.

Each of the 27 features from the feature model was specified according to the feature specification template. In addition, during the Scoping activity, a list of products for the mobile application for the emergency notifications domain was defined, thus allowing the creation of the Product Map artifact. With regard to the Requirements Specification for Domain Engineering activity, two requirements analysts from the team created the Glossary artifact based on the artifacts that had been created in the Scoping activity. A total of 16 relevant terms were identified for the domain. An excerpt of this artifact is shown in Table 5.4.

Table 5.4: Excerpt from the Glossary.

Term	Definition
Contact	It represents a person to be contacted in an emergency situation. It includes relevant information like e-mail, phone number, Facebook ID, Twitter ID.
Contact List	Collection of contacts sorted in an alphabetical order.
Twitter	Micro blogging service. It is a site on which the user can share small messages and retrieve contacts to SAVi.
User	It represents the person who uses the application.

⁴Help.me: <http://goo.gl/hSWpq> | Rescue Button: <http://goo.gl/asli3> | Red Panic Button: <http://goo.gl/FpVsk> | RescueMe Now: <http://goo.gl/pDY9o>



Figure 5.8: Selected Features from the Feature Model for the Case Study.

The artifacts created by the Scoping activity (Feature Model, Feature Specification and Product Map) and the Glossary artifact created by the Requirements Specification for Domain Engineering activity made it possible to create the Functional Requirements and the Traceability Matrix artifacts by applying the guidelines for specifying SPL requirements.

5.4.3 Collection of the data

The data for this empirical study was collected during the Requirements Specification for Domain Engineering activity. The SPL Functional Requirements were specified by recruiting fourteen Ph.D. students, from both universities (Spain and Brazil), who were asked to apply the guidelines for specifying SPL functional requirements (shown in Sub-Section 5.3.3) in order to answer the following questions: i) **Which** features can be grouped to be specified by Use Cases (UC)?; ii) **What** are the specific use cases for the feature or set of features?; iii) **Where** should the use case be specified?; and iv) **How** is each use case specified in terms of steps? These subjects had a background in SPL and half of them have a background in SPL RE.

5.4.3.1 Which features can be grouped to be specified by UC?

This step analyzes all the features included in the increment unit for the current iteration. The subjects had to decide which of these features (see Figure 5.8) would be specified by use cases. According to the first task of the guidelines, most of the requirements analysts (subjects) started the iteration with the feature *Access_Control* and its children, because they are a group of features that share functionality (Task 1 from the guidelines). Since there are two ways of implementing an import contact (one optional: *Web_Access_Control*; and one mandatory: *Mobile_Access_Control*) some requirements analysts followed the guidelines (Steps 1.2 and 1.4 from the guidelines) and decided that those features would not be specified as use cases. Thus, they were specified as alternative scenarios in the use case related to the feature *Access_Control*. In a similar way, some subjects specified the features *Facebook_Import*, *Twitter_Import* as alternative scenarios from a use case of the *Import_Contact* feature, and some subjects specified the features *SMS_Destination*, *Twitter_Destination*, *Facebook_Destination* and *Email_Destination* as alternative scenarios in use cases related to the *Destination* feature. Following the guidelines, most of the subjects decided that the features: *Contact*, *Import_Contact*, *Add_Contact*, *Destination* and *Emergency_Numbers* would be specified as use cases. Unfortunately, there were some subjects that decide not to specify alternative scenarios as the guidelines recommend, ignoring the variability from the feature model.

5.4.3.2 What are the specific UC for the feature or set of features?

After deciding which features need to be specified as use cases, the subjects had to identify which use cases should be associated to each feature. Moreover, the Traceability Matrix was incrementally filled in with traceability information between the use case and the feature. An excerpt of the traceability matrix (features X UC) is shown in Table 5.5.

Table 5.5: Excerpt from the Traceability Matrix.

	UC012	...
Access_Control	X	
Mobile_Access_Control	X	
Web_Access_Control	X	

The most common identified use cases by the subjects for the selected features are presented next⁵. The following use cases were identified for the *Access_Control* feature: *Create_User*, *Login*, *Show_Profile*, *Remember_Password* and *Send_E-mail*. The following use cases were identified for the *Web_Access_Control* feature: *Update_User*, and *Delete_User*. The following use cases were identified for the *Contact* feature: *Show_Contact*, *Delete_Contact* and, *Update_Contact*. The following use case was identified for the *Add_Contact* feature: *Add_Contact*. The following use cases were identified for the *Import_Contact* feature: *Retrieve_Contacts* and *Import_Contacts*. The *Destination* feature contains the use case *Send_Notification*. The following use cases were identified for *Emergency_Numbers* feature: *Create_Emergency_Number*, *Delete_Emergency_Number* and *Update_Emergency_Number*. The corrected number of use cases to be identified by the subjects should be seventeen use cases for the selected features (Figure 5.8).

5.4.3.3 Where the UC should be specified?

Since some use cases with similar behavior may be identified for different features that have the same parent, the subjects should decide where to relocate the specification for this use case (this is to avoid the redundant specification of similar behavior). When this happens, the use case was specified once only at the parent feature level. As soon as all the use cases have been identified for each feature, it is possible to start modeling the use cases. A use case package is created for each feature that will have use cases, and a use case diagram is created to include the use cases, actors and relationships among them. An example for the *Access_Control* feature (use case diagram) is shown in Figure 5.9.

⁵An additional table with the list of identified Use Cases for each Feature is available in Appendix C.2

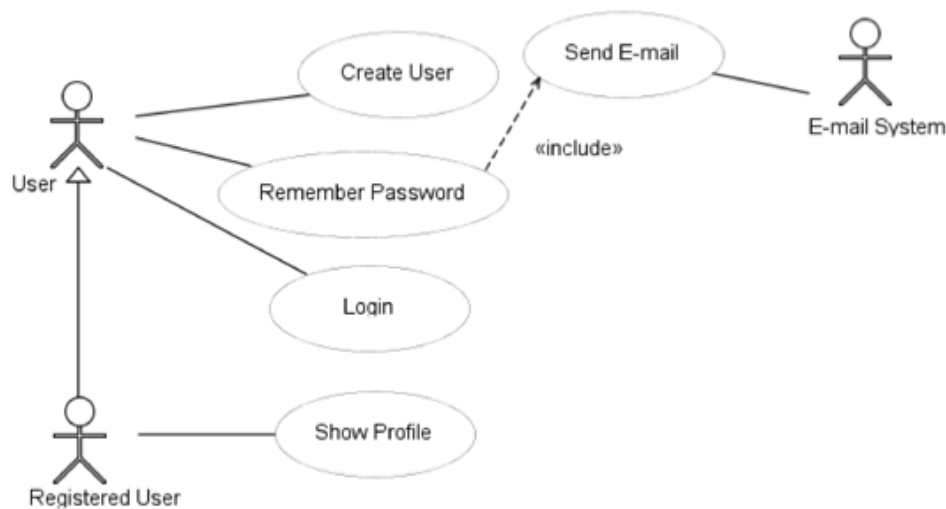


Figure 5.9: UC Diagram (Feature Access Control).

5.4.3.4 How each UC is specified in terms of steps?

After identifying the use cases and relating them to the features, the subjects specified each use case by taking into account the variations from the Feature Model. Table 5.6 shows the Functional Requirement specification for one of use case related to the *Access_Control* feature, which is the *Login* use case.

This use case specification has the optional feature *Web_Access_Control*, which is specified as an alternative scenario. This use case specification thus handles the variability expressed in the Feature Model in which, depending on the selected feature, an alternative scenario can be included in the use case specification. Another advantage of using alternative scenarios to represent the variability is that of reuse. Since the alternative scenarios are specified once only within a use case, several products can be instantiated by reusing the same use case. Different behaviors may thus appear in the same use case for different products, depending on the selected features.

5.4.4 Data Analysis

It was performed two quantitative analyses for the collected data. The first analysis was related to the objective variables (effectiveness, efficiency) observed during the execution of both tasks. Since we did not have a control group, this analysis was performed in order to identify deficiencies in the guidelines and improve the redaction in a qualitative manner. The second quantitative analysis was performed by using closed questions, which were filled in by the subjects after the empirical study execution. This information was analyzed in a quantitative

Table 5.6: Retrieve Contacts Use Case Specification.

*Use case id:	UC012		
*Name:	Login		
*Description:	It allows a registered user to access the system		
Associated feature:	Access_Control	Actor(s) [0..]:	User
*Pre-condition:	The User is not logged in	*Post-condition:	The User accesses the system
Includes To:	-	Extends From:	-
*Main Success Scenario			
Step	Actor Action	Blackbox System Response	
1	The user asks to login using the mobile	The System shows the username and password fields to be filled in	
2	The User fills in the username and password fields	The System validates the username and password and allows the user access	
Alternative Scenario name:		Web Access Control Login	
Condition:		The user must be using a computer	
Associated feature [0..1]:		Web Access Control	
Step	Actor Action	Blackbox System Response	
2.1	Requires the login through a computer	The System shows a form to be filled in	
2.2	The User fills in the username and password and confirm	The System login in to Savi by using the Web Access for computers	

* Mandatory Field

manner in order to check the results of the subjects' perception of ease of use and usefulness, and their statistical relevance.

5.4.4.1 First Quantitative Analysis

Regarding effectiveness in task 1 of the empirical study, we measured the quotient of the right number of uses cases identified by the total number of use cases modeled in the solution (*Effectiveness_UC*). The users were able to identify correctly 62.1% of the total use cases in the task 1. In order to check the effectiveness in task 2 of the empirical study, we compared the specified alternative scenarios with the scenarios modeled in the solution (*Effectiveness_SCEN*). The results show that this variable has a mean of 0.523, meaning that the users were able to correctly specify 52.3% of the total use cases (see Table 5.7).

Additionally, for each task we measured the time used and the efficiency estimation. The results show that the subjects took around 51 minutes to complete the task 1, with an *Efficiency_UC* value of 0.216 (number right use cases / time). The subjects took around 39 minutes to complete the task 2, with an *Efficiency_SCEN* value of 0.048 (number of right alternative scenarios / time). We believe that this time for identifying the use cases and their alternative

scenarios is still high. A tool support may decrease the overall time spent in these activities.

Table 5.7: Mean and Standard Deviation for the analyzed variables.

	Mean	SD*
Effectiveness_UC	0.621	0.182
Effectiveness_SCEN	0.523	0.447
Time task 1	51.86	13.091
Efficiency_UC	0.216	0.085
Time task 2	39	19.247
Efficiency_SCEN	0.0484	0.048

*SD: Standard Deviation

Finally, a qualitative analysis was performed by analyzing the open questions that were included in the questionnaire. For example, some subjects suggested reformulate some guideline rules to avoid ambiguities during the use cases identification (*e.g.*, identifying group of features that share functionality), or during the use case specification (*e.g.*, defining alternative scenarios). Other subjects suggested including in the guidelines rules for dealing with relationships among features (includes / extends). The analysis of these qualitative data revealed several important issues that have to be considered to improve FeDRE.

5.4.4.2 Second Quantitative Analysis

In this section, it is discussed the results of the empirical study by quantitatively analyzing the data according to the hypotheses stated. All the results presented were obtained by using the SPSS v20 statistical tool with an alpha value of 0.05. The subjective variables were analyzed by comparing whether the mean of the responses to the questions related to each variable were significantly greater than the Likert neutral value⁶ (equal to 3). In our case, the mean variable ranging from 1 to 5 for the measurement of both subjective variables has been considered as an interval scale (Carifio and Perla, 2007). Both variables have a mean over the neutral value 3 (see Table 5.8)⁷.

In order to verify the hypotheses with this data, we first selected which test was most appropriate for the data. It was first necessary to check whether the data was normally distributed. Since the sample size is smaller than 50, we applied the Shapiro-Wilk test to verify whether the data is normally distributed. The results of the normality test (Table 5.8) show that both variables are normally distributed in this evaluation method since the results are greater than 0.05.

⁶The subjects' responses are available in Appendix C.3.

⁷The box plots for the subjective PEOU and PU variables are shown in Appendix C.4

As consequence, we check the hypotheses by performing a one-tailed t-test for independent variables with a test value of 3. The p-values obtained (Table 5.8) were < 0.05 ($p \leq \alpha$). As consequence we reject both null hypotheses; accepting that FeDRE is perceived as easy to use and useful.

Table 5.8: Analysis of the PEOU and PU variables.

	Mean	SD	Shapiro-Wilk	Alpha Cronbach	t-test*
Perceived Ease of Use	3.857	0.813	0.696	0.833	0.002
Perceived Usefulness	3.880	0.771	0.066	0.722	0.001

* P-values from the one-tailed t-test

5.4.5 Threats to validity

The main threats to the **internal validity** of the empirical study are: empirical study design, evaluation design, subject experience, information exchange among evaluators, and the understandability of the documents. It was selected the *empirical study design* based on the methods proposed by [Abrahão et al. \(2011\)](#) and [Davis \(1989\)](#) (TAM) to evaluate FeDRE. There are some drawbacks related to TAM, as presented by [Turner et al. \(2008\)](#). However, since there is no approach with similar objectives such as the ones from FeDRE, it was used these methods ([Davis, 1989](#); [Abrahão et al., 2011](#)) to evaluate FeDRE. It is important to highlight that these methods are still been used nowadays to evaluate new proposed approaches ([Oliveira et al., 2013](#); [Fernandez et al., 2013](#); [Molina et al., 2013, 2014](#); [Asadi et al., 2015](#)). The *evaluation design* can have affected the results owing to the selection of features to be taken as input to extract the requirements when applying FeDRE. We attempted to alleviate this threat by considering a subset of features, which implied applying the complete set of the FeDRE guidelines rules. *Subject experience* was alleviated owing to the fact that none of the subjects had any experience in requirements modeling in SPL development. *Information exchange* was mitigated by monitoring the participants while they performed the tasks. We performed the experiment in two sessions but no relationships were established between Spanish and Brazilian subjects and no information was exchanged among them. We alleviated the *understandability of the material* by clearing up all the misunderstandings that appeared in each session.

The main threat to the **external validity** of the experiment is the *representativeness of the results*. To alleviate this threat and make the tasks enough representative, the complexity of the exercise was adjusted for the subjects to be able to apply every single rule of the guidelines at least once, considering that the duration of the experiment was limited to 90 minutes.

The main threat to the **construct validity** of the experiment was the reliability of the questionnaire, related to the two empirical study hypotheses. This *reliability* was tested by applying the Cronbach's alpha test to each set of closed questions which measured the PEOU and PU variables, obtaining a value of 0.833, and 0.722 respectively (higher than the minimum acceptance threshold $\alpha=0.70$) (Maxwell, 2002).

The main threat to the **conclusion validity** of the experiment was the *pattern recognition (recall and precision) for the effectiveness and efficiency of FeDRE activities*, and the *validity of the statistical test applied*. For this empirical study we used only the *recall pattern to evaluate the effectiveness and efficiency of FeDRE activities*, however, this is a well know and used pattern. The *validity of the statistical test applied* was alleviated by applying the most common test that is employed in the empirical software engineering field (Maxwell, 2002).

5.5 Chapter Summary

This Chapter introduced the FeDRE approach to support the requirements specification of SPL. In this approach, chunks of features from a feature model are realized into functional requirements, which are then specified by use cases. The required requirements variations can be related to the use case as a whole or to alternative scenarios inside a use case. A set of guidelines was provided to help SPL developers to perform these activities and as a means to systematize the process and ensure a correct traceability between the different requirements artifacts. We believe that this approach provides a solution that is capable of dealing with the complexity involved in SPLs with a large number of requirements and features.

The feasibility of FeDRE was evaluated using an empirical study involving a mobile application for emergency notifications. The results show that the analysts perceived the approach as easy to use and useful for specifying the functional requirements in this particular SPL. However, the approach needs further empirical evaluation with larger and more complex SPLs. Moreover, guidelines for dealing with the evolution of the specification created by FeDRE are needed, since there is a need for evolving the requirements according to the user needs and the environment.

The evolution of the requirements specification created by FeDRE is presented in the next Chapter.

6

Feature-Driven Requirements Engineering Evolution (FeDRE²) Approach

This Chapter presents the *Feature-Driven Requirements Engineering Evolution (FeDRE²)* approach, which is an approach for dealing with the evolution of SPL requirements specified by the FeDRE approach.

The Chapter is organized as follows: Section 6.1 presents an introduction to SPL evolution and shows the motivation to propose this approach. Section 6.2 presents the background concepts and the related work. Section 6.3 describes FeDRE² approach for evolving SPL requirements in a systematic way, through guidelines. Section 6.4 presents the performed empirical study, including the defined design, the preparation of the study, the data collection, the analysis, and threats to validity of this work. Finally, Section 6.5 presents the Chapter summary.

6.1 Introduction

Clements and Northrop (2002) firstly pointed out that the nature of an SPL is to manage the commonality and variability of products by means of a “*Requirements Engineering (RE) – change management*” process. Within their book (Clements and Northrop, 2002), they highlighted the importance of RE change management for SPL. However, since their book covers the whole SPL process in general, they did not discuss in details how the RE change management is performed in SPL.

Later, other work tried to improve the evolution of features (Ye and Liu, 2005; Gomaa, 2013; White *et al.*, 2014) and use cases (Gomaa and Shin, 2008) in SPL. Ye and Liu (2005) deal with the evolution of features and their approach can detect a conflict in a feature dependency. Gomaa (2013) proposed an evolutionary development approach where feature modeling and

SPL concepts are used to evolve software requirements and architectures. [White et al. \(2014\)](#) presented a systematic approach for evolving feature models and performed an experiment to evaluate their approach. [Gomaa and Shin \(2008\)](#) proposed a multiple-view meta-modeling approach for SPL, using the Unified Modeling Language (UML), which allows the evolution of an SPL by explicitly modeling the variation points. However, none of these approaches perform the evolution of features and use cases in a systematic way, through the use of guidelines. We claim that the use of guidelines can make SPL RE evolution process easier and more effective.

[Alves et al. \(2010\)](#) performed a systematic literature review on SPL Requirements Engineering (RE). They discovered that the majority of the papers relies on a low level of evidence and evaluation (*i.e.*, using toy examples) and also, the majority of the papers lack of guidance for performing the requirements engineering activities within SPL.

SPL artifacts must evolve ([Oliveira et al., 2015a](#)), mainly the requirements. Requirements are considered the first artifacts from an SPL project, if they do not support well the evolution, the artifacts from the following activities (architecture, implementation, test, and so on) can be outdated, making harder the creation of new SPL products. Moreover, since the user needs within SPL are represented through requirements, they also must have the ability to evolve.

Thus, this Chapter presents the *Feature-Driven Requirements Engineering Evolution (FeDRE²)* approach to evolve SPL requirements in a systematic way, using guidelines, driven by features. Based on a change request, the requirements engineer selects one of the FeDRE²'s evolution scenarios and performs the evolution of the SPL requirement according to a set of guidelines, including updating the traceability matrix. Since there is a lack of work dealing with the SPL requirements evolution in a systematic way, there is also a lack of empirical evaluation in this area ([Alves et al., 2010](#)). Hence, it was conducted an empirical study, composed of two sessions, to evaluate the proposed approach. Both sessions were performed by post graduate subjects (with Bachelor, Master, and Ph.D. degrees in Computer Science). The first session was performed with Brazilian students from Federal University of Bahia and the second one was performed with Spanish students from Polytechnic University of Valencia.

6.2 Background

Earlier in Chapter 5, it was presented the approach for specifying the SPL requirements, called *Feature-Driven Requirements Engineering (FeDRE)* ([Oliveira et al., 2014](#)). However, FeDRE is responsible only for *creating* features, use cases, and the traceability matrix. FeDRE does not deal with their *evolution*. Thus, this Chapter proposes *Feature-Driven Requirements Engineering*

Evolution (FeDRE²) approach, which uses some guidelines of FeDRE, and it also has new guidelines for evolving the SPL features, use cases, and the traceability matrix.

FeDRE² guidelines use the *Safe Evolution Templates*, proposed by Neves *et al.* (2011). They defined a total of 8 templates and they were evaluated by analyzing the evolution history of two different SPL. Neves *et al.* also demonstrated that the safe evolution templates can express the corresponding modifications and may help to avoid the mistakes during the SPL evolution. Thus, FeDRE² adapted the safe evolution templates for dealing with features and use cases. Moreover, FeDRE² approach identified two new safe evolution templates that may be used to evolve features and use cases.

FeDRE² overview and its guidelines were also built based on the *Software and Systems Process Engineering Meta-model (SPEM)*¹. Figure 6.1 shows the SPEM profiles used by FeDRE².

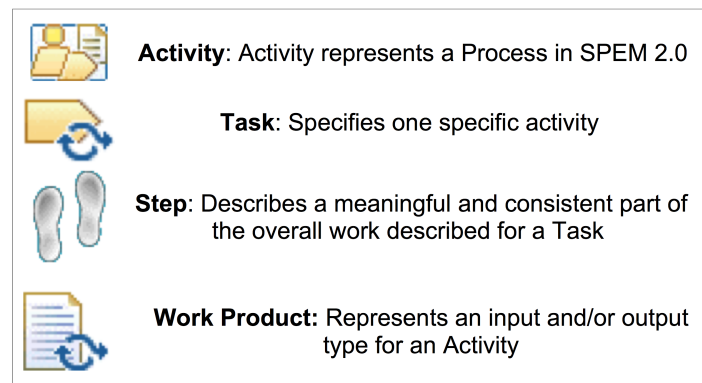


Figure 6.1: SPEM Profiles used by FeDRE²

6.2.1 Related Work

To the best of our knowledge, FeDRE² is the only approach for dealing with the evolution of SPL features and use cases conducted by guidelines in a systematic way. Moreover, FeDRE² approach was evaluated through a rigorous empirical study. However, there are other approaches dealing with the evolution of SPL features and/or use cases.

¹<http://www.omg.org/spec/SPEM/2.0/>

[Gomaa and Shin \(2008\)](#) proposed a multiple-view meta-modeling approach for SPL using the Unified Modeling Language (UML). This meta-modeling defines several SPL artifacts, such as: use case model; static model (class diagrams); collaboration model; state-chart model; and feature model, including the commonality and variability. They state that a strong point of the multiple-view modeling approach is to allow the evolution of SPL by explicitly modeling the variation points in each view where evolution can take place and by defining the relationships between these variation points. Nonetheless, they deal with low level of SPL requirements evolution (meta-modeling) and evaluated their approach through a proof of concept tool.

The approach proposed by [Ye and Liu \(2005\)](#) focuses on modeling variability and dependencies, within the feature model, through dependency view. A matrix and a set of dependency diagrams is used to accommodate the feature dependencies. Their approach can deal with the evolution of SPL features and it also can detect a conflict in a feature dependency. However, besides dealing only with the evolution of features, the approach was evaluated through a feasibility study.

[White et al. \(2014\)](#) presented a systematic approach for evolving feature models through a multi-step configuration. They claim that a multi-step is necessary because an evolution may need to be broken in multiple steps to satisfy evolution constraints. Also, they present how a multi-step configuration can be mapped to a Constraint Satisfaction Problem (CSP). They evaluated their approach through two experiments. Besides all the systematization within the proposed approach, they only deal with feature models.

[Gomaa \(2013\)](#) proposed an evolutionary development approach where feature modeling and SPL concepts are used to evolve software requirements and architectures. He considers as an SPL the different versions of a software systems, and he tried to understand how feature modeling can be used to characterize variability due to requirements evolution and what is the impact in the architecture. However, in order to evolve the software requirements, he does not propose any kind of guidelines. Moreover, the approach was not evaluated.

As shown previously, there is still a lack of approaches that propose guidelines for performing the SPL evolution of features and use cases in a systematic way. The current approaches do not present clear steps for performing the evolution of features and use cases in a systematic way. Moreover, most of the presented approaches were not empirically evaluated. Thus, FeDRE² was proposed and evaluated following the guidelines presented by [Wohlin et al. \(2012\)](#), where statistical tests could be applied to evaluate the hypotheses.

6.3 FeDRE² Approach

In order to safely evolve the SPL requirements, it was proposed the *Feature-Driven Requirements Engineering Evolution (FeDRE²)* approach. This approach considers the requirements specified by FeDRE approach and deals with their evolution. FeDRE² is an approach for SPL domain engineering which deals with the evolution of features and requirements. The aim of the approach is to perform the requirements evolution by systematically utilizing the features identified in the SPL domain through the use of guidelines that establish traceability links between features and requirements, represented herein by use cases². An overview of the approach, using SPEM, is shown in Figure 6.2.

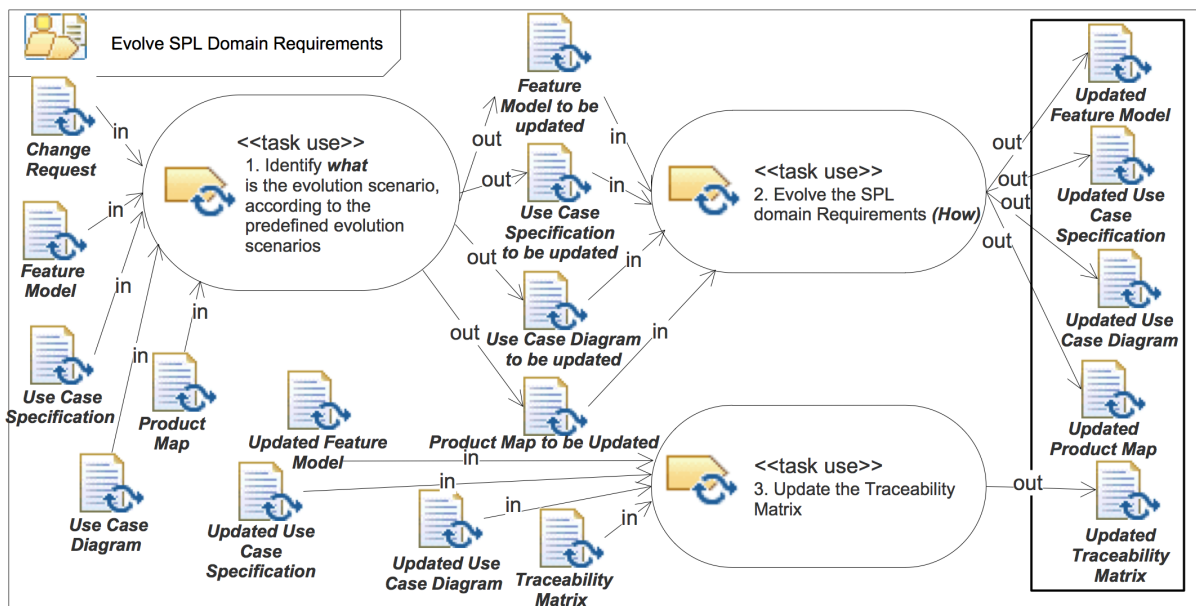


Figure 6.2: FeDRE² Approach Overview.

The approach is composed of three main tasks: 1) *Identify the evolution scenario (what)*; 2) *Evolve the SPL requirements (how)*; and 3) *Update the traceability matrix*. Each one of the activities has a set of steps to be followed according to the guidelines. These tasks are defined in FeDRE², not in FeDRE. However, task 2 (Evolve the SPL requirements) may invoke FeDRE approach depending on the sub-task to be performed. The tasks are detailed next.

²The metamodel representing the relationships among features and use cases was presented in Figure 5.5

6.3.1 Task 1: Identify the Evolution Scenario

Within this first task of FeDRE² approach, the requirements engineer has to identify what is the evolution scenario. The evolution scenarios supported by FeDRE² approach were defined based on the safe evolution templates for SPL proposed by Neves *et al.* (2011). Table 6.1 shows the safe evolution templates proposed by Neves *et al.* and also their relationship with the FeDRE² evolution scenarios.

Table 6.1: Safe Evolution Templates, Evolution Scenarios and Evolution Scenarios Description.

Safe Evolution Template (Neves <i>et al.</i> , 2011)	FeDRE ² Evolution Scenario	Evolution Scenario Description
Split Asset	1.1. Split an Use Case	Split the use case alternative scenarios into new use cases
Refine Asset	1.2. Refine an Use Case	Improve the use case steps (main scenario and/or alternative scenarios)
Add New Optional Feature	1.3. Add New Optional Feature	Add a new optional feature into the feature model
Add New Mandatory Feature	1.4. Add New Mandatory Feature	Add a new mandatory feature into the feature model
Replace a Feature Expression	1.5. Replace a Feature Expression	Update the relationship between the feature and the use case in the traceability matrix
Add New Alternative Feature	1.6. Add New Alternative (XOR) Feature	Add a new alternative (XOR) feature into the feature model
Add New OR Feature	1.7. Add New Alternative (OR) Feature	Add a new alternative (OR) feature into the feature model
Delete Asset	1.8. Deletion	Delete a feature, or an use case, or an use case alternative scenario
-	1.9. Transform a Mandatory Feature into an Optional Feature	Transform a mandatory feature into an optional feature in the feature model
-	1.10. Transform an Optional Feature into a Mandatory Feature	Transform an optional feature into a mandatory feature in the feature model

To identify which is the evolution scenario, the requirements engineer needs the input artifacts of this task: the change request; the feature model; the use case textual specification; the use case diagram specification; and the product map.

6.3.1.1 Change Request Artifact

The *Change Request (CR)* artifact is responsible for registering the evolution requisition according to the users' need. An example of such artifact is shown in Figure 6.3.

When registering a CR, the user should fill in these main fields: the *Summary*, which is the CR brief description; *Description*, which contains the full description of the CR; the *Type*, which indicates if the CR is an enhancement, an error, or a suggestion; the *Priority*, which could be minor, normal, or major; the *Document*, which indicates what kind of document the CR is related to, for example the feature model document or the requirement document; and the

Related Feature indicating to which feature the CR is related to. These are the main fields in the CR that need to be filled in. The next artifact of the approach to be presented is the feature model. Each CR has a unique identification (ID), which is called *ticket* (not visible to the end user).

Properties

Summary: Upload a photo for the user profile

Reporter: raphaeloliveira

Description: **B I A** You may use WikiFormatting here.
 Allow the user to upload a photo for his/her profile. In this case, the system should ask for a photo (optional) during the edition screen.

Type: enhancement

Milestone: 2.0

Version:

Cc:

Related Feature: 25-Web Access Control

Priority: major

Component: Client-side component

Keywords:

Document: feature

Owner: < default >

Figure 6.3: Change Request Example.

6.3.1.2 Feature Model Artifact

As shown in Section 5.3.1.2 and Section 5.3.1.3, the *feature model* artifact comprises the features of the domain, being responsible for identifying features, SPL variations, and constraints among the features of the SPL. An example of a feature model artifact is shown in Figure 6.4.

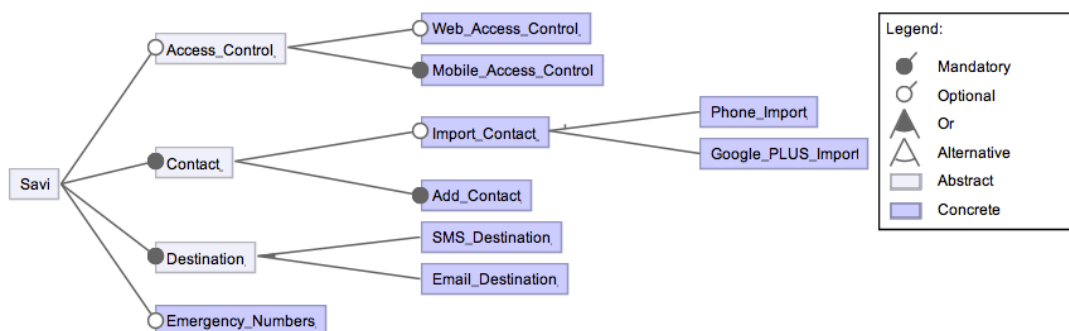


Figure 6.4: Feature Model Example.

6.3.1.3 Use Case Textual Specification Artifact

The *use case textual specification* contains the textual use cases specified using FeDRE approach. Its description was previously presented in Section 5.3.2.2. An example of such artifact representing two variations from an OR relationship of the feature model (Figure 6.4) is shown in Table 6.2.

Table 6.2: Send Message Use Case Textual Specification Example.

*Use case id:	UC015		
*Name:	Send Message		
*Description:	It sends a message to the user's contacts		
Associated feature:	Destination	Actor(s) [0..]:	Logged User
*Pre-condition:	The logged user must press the Savi button	*Post-condition:	The system sends a message to the contacts of the user
Includes To:	-	Extends From:	-
*Main Success Scenario			
Step	Actor Action	Blackbox System Response	
1	The Logged user press the Savi button	The system sends a message to the contacts of the user	
Alternative Scenario name:		SMS Message	
Condition:		An SMS message should be sent to the user's contact list	
Associated feature [0..1]:		SMS_Destination	
Step	Actor Action	Blackbox System Response	
1.1	-	The system sends an SMS message to the user's contact list	
Alternative Scenario name:		Email Message	
Condition:		An Email message should be sent to the user's contact list	
Associated feature [0..1]:		Email_Destination	
Step	Actor Action	Blackbox System Response	
1.1	-	The system sends an Email message to the user's contact list	

* Mandatory Field

6.3.1.4 Use Case Diagram Artifact

The *use case diagram* is the artifact that contains the use case elements, such as actors, use cases, and their relationships. An example of such artifact is shown in Figure 6.5.

6.3.1.5 Product Map Artifact

Finally, the *product map* is the artifact that contains the relationships among features and the products from the SPL. An example of such artifact is shown in Figure 6.6.

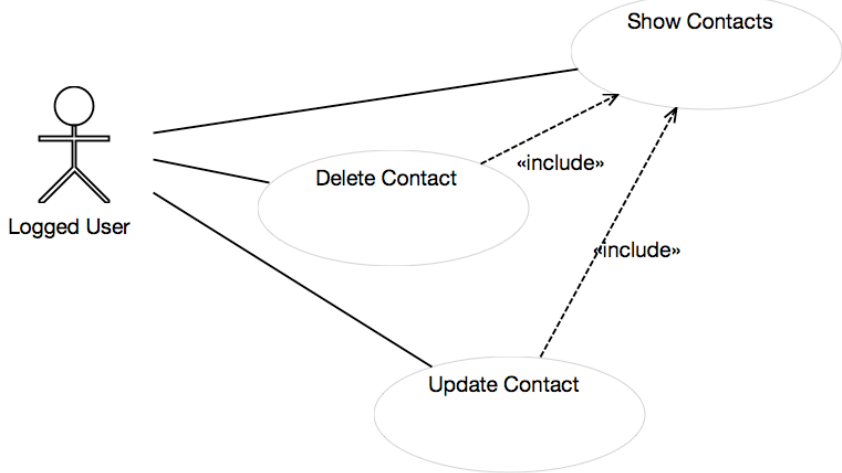


Figure 6.5: Use Case Diagram Example.

Features / Products	Savi Lite	Savi Standard	Savi Social	Savi Pro	Savi Ultimate
Access_Control			X	X	X
Web_Access_Control			X	X	X
Mobile_Access_Control			X	X	X
Contact	X	X	X	X	X
Add_Contact	X	X	X	X	X
Import_Contact		X	X	X	X
Facebook_Import			X	X	X
Twitter_Import			X	X	X
Phone_Import		X	X	X	X
Destination	X	X	X	X	X
SMS_Destination	X	X	X	X	X
Twitter_Destination			X	X	X
Facebook_Destination			X	X	X
Emergency_Numbers		X	X	X	X

Figure 6.6: Product Map Example.

With the input artifacts, the requirements engineer will follow the steps from sub-task number 1 of the FeDRE² guidelines (Figure 6.7) to identify the evolution scenario. Once identified the evolution scenario, the requirements engineer has to know what should be updated within the feature model, the use case specification, the use case diagram artifacts, and the product map.

6.3.2 Task 2: Evolve the SPL Requirements

The next task of the approach consists of evolving the SPL requirements. To evolve the feature model and use case (textual and diagram specifications), the requirements engineer has as input following artifacts to be updated: feature model; use case specification, the use case diagram, and the product map.

The requirements engineer needs to follow once again the FeDRE² guidelines (Figure 6.7) to evolve the requirements. The sub-tasks starting with the number 2 within the guidelines are responsible for performing the SPL requirement evolution. There are 10 sub-tasks for evolving SPL requirements. Each one has a set of steps to evolve the SPL requirements according to the identified evolution scenario, as follows:

- *2.1. Split a Requirement:* First, it is necessary to confirm that the use case to be split has at least 1 alternative scenario, and each alternative scenario must be associated to a feature. Then, for each alternative scenario to be split, the requirements engineer will create a new optional use case, following FeDRE guidelines (Figure 5.7). Finally, the alternative scenarios from the split use case must be removed.

- *2.2. Refine a Requirement:* The requirements engineer needs to select the steps in the main scenario or in the alternative scenario to be improved and perform the update.

- *2.3. Add a New Optional Feature:* It is necessary to start adding the new optional feature into the feature model. After this step, the requirements engineer needs to follow the FeDRE guidelines for specifying the new optional use case(s) or the new alternative scenario(s). The requirements engineer also need to update the product map according to the new optional feature.

- *2.4. Add a New Mandatory Feature:* The first step is to add the new mandatory feature into the feature model. Afterwards, the requirements engineer needs to follow the FeDRE guidelines for specifying the new mandatory use case(s). The requirements engineer also need to update the product map according to the new mandatory feature.

- *2.5. Replace a Feature Expression:* First, the requirements engineer needs to update the feature associated to the use case. Then, it is required to follow FeDRE guidelines to update the requirement(s) (diagrams and textual specifications).

- *2.6. Add New Alternative (XOR) Feature:* It is necessary to add the new alternative (XOR)

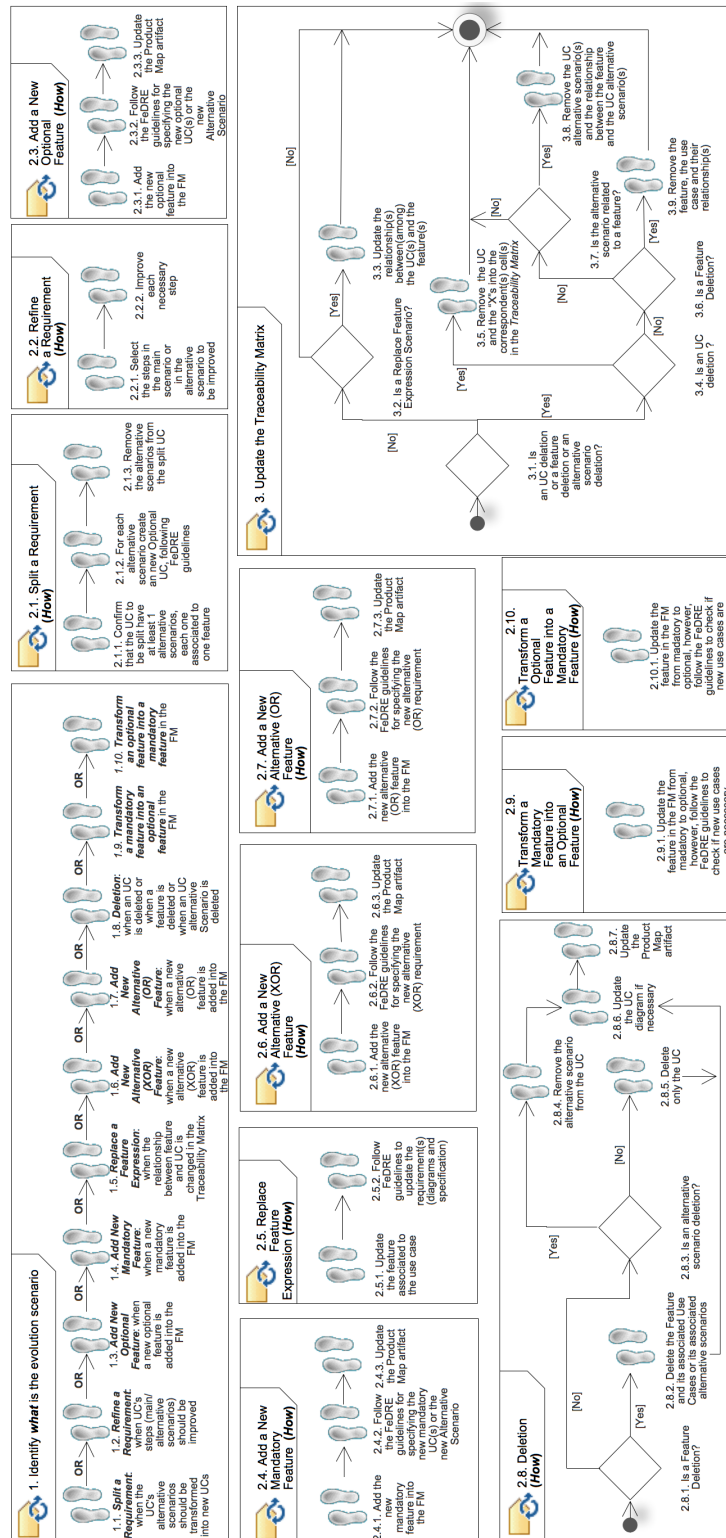


Figure 6.7: Guidelines For Evolving SPL Functional Requirements.

feature into the feature model and then follow the FeDRE guidelines for specifying the new alternative (XOR) use case(s). The requirements engineer also need to update the product map according to the new alternative (XOR) feature.

- 2.7. *Add New Alternative (OR) Feature*: The requirements engineer should add the new alternative (OR) feature into the feature model and follow the FeDRE guidelines for specifying the new alternative (OR) use case(s). The requirements engineer also need to update the product map according to the new alternative (OR) feature.

- 2.8. *Deletion*: Firstly, the requirements engineer needs to check if the evolution is a feature deletion. If so, it is necessary to delete the feature and its associated use case(s) or its associated alternative scenario(s). If the evolution is the deletion of an alternative scenario, the requirements engineer needs to remove the alternative scenario from the use case. Finally, if the evolution is the deletion of an use case, the requirements engineer must delete only the use case. After each one of these evolutions, if necessary, the use case diagram and the product map should be updated.

- 2.9. *Transform a Mandatory Feature into an Optional Feature*: Within this evolution, the requirements engineer needs to update the feature in the feature model from mandatory to optional. However, it is necessary to follow the FeDRE guidelines to check if new use cases are necessary.

- 2.10. *Transform an Optional Feature into a Mandatory Feature*: It is necessary when the requirements engineer updates the feature in the feature model from optional to mandatory. However, it is necessary to follow the FeDRE guidelines to check if new use cases are necessary.

Thus, after performing the evolution within the SPL requirements, the outputs of this task will be the following artifacts: the updated feature model; the updated use case specification; the updated use case diagram; and the product map.

6.3.3 Task 3: Update the Traceability Matrix

The final task of the approach is to update the traceability matrix, which stores the relationships among features and requirements.

6.3.3.1 Traceability Matrix Artifact

The traceability matrix, which stores the relationships among features and requirements, and was presented previously in Section 5.3.2.3, is shown in Figure 6.8.

To update the traceability matrix, the requirements engineer has as input the following artifacts: the updated feature model; the updated use case specification; the updated use case

Use Cases \ Features	Access_Control	- Web_Access_Control (Alternative Scenario from Show Profile)	- Mobile_Access_Control (Alternative Scenario from Show Profile)	Contact	Add_Contact	Import_Contact
Create User	X					
Login	X					
Show Profile	X	X	X			
Remember Password	X					
Send E-mail	X					
Update User Profile		X				
Delete User		X				
Show Contacts				X		
Delete Contact				X		
Update Contact				X		
Show Message Language					X	
Add Contact					X	
Retrieve Contacts						X
Import_Contact						X

Figure 6.8: Excerpt of a Traceability Matrix.

diagram; and the traceability matrix.

The requirements engineer needs to follow the FeDRE² guidelines (Figure 6.7) to update the traceability matrix, which steps are inside the sub-task number 3. The main questions that the requirement engineer should answer is if the evolution is a feature deletion, or an use case deletion, or an use case alternative scenario deletion.

If the evolution is an feature deletion, the requirements engineer needs to remove from the traceability matrix the feature, the use case(s) associated to the feature, and their relationship(s). If the evolution is a deletion of an use case alternative scenario, the requirements engineer has to remove from the traceability matrix the relationship between the feature and the use case alternative scenario(s). If the evolution is an use case deletion, the requirements engineer should remove from the traceability matrix the use case and its relationships. Otherwise, if the evolution is a replace feature expression scenario, the requirements engineer needs to update the relationship between (among) the use case(s) and the feature(s). After updating the traceability

matrix, the SPL requirement evolution is completed. Next, it is presented the empirical study performed to evaluate FeDRE² approach.

6.4 Empirical Study

It was performed an empirical study, following the guidelines presented by Wohlin *et al.* (2012), to assess the easy of use and the usefulness of FeDRE² approach. Besides this is its first evaluation, the obtained results have shown FeDRE² as a promising approach.

6.4.1 Design of the Empirical Study

In order to evaluate FeDRE², we firstly planned the objectives of our empirical study. To define which objectives the empirical study would have, it was applied the *Goal-Question-Metric (GQM)*. After applying this technique, it was stated the goal of the empirical study, as follows: to **analyze FeDRE² for the purpose** of evaluating it **with regard to** its perceived ease of use and perceived usefulness **from the viewpoint** of a set requirements engineers **in the context** of an SPL project.

The context of the empirical study is the requirements evolution of an SPL project. The selected SPL, for the mobile domain, was the same used in FeDRE evaluation, SAVi SPL.

It was defined two subjective dependent variables: *perceived ease of use* and *perceived usefulness*. To measure both variables after applying the FeDRE² approach, it was used an existing measurement instrument proposed for the evaluation of requirements modeling methods based on user perceptions (Abrahão *et al.*, 2011). The method proposed by Abrahão *et al.* (2011) allows to evaluate the likelihood of acceptance of a particular method in practice or to compare two or more requirements modeling methods to assess which is the most effective. Since there is no such approach as FeDRE², we used the method to evaluate the likelihood of acceptance of FeDRE², without comparing it to another approach. More specifically, we adapted two perception-based variables from the aforementioned instrument, which were based on two constructs from the *Technology Acceptance Model (TAM)* (Davis, 1989):

- **Perceived Ease of Use (PEOU)**: the degree to which a person believes that using FeDRE² will be effort-free. This variable represents a perceptual judgment of the effort required to learn and use the FeDRE² approach;

- **Perceived Usefulness (PU)**: the degree to which a person believes that FeDRE² will achieve its intended objectives. This variable represents a perceptual judgment of the FeDRE² approach's effectiveness.

It was defined a set of items to measure these perception-based variables. These items were combined in a survey consisting of 10 statements. The items were formulated by using a 5-point Likert scale, using the opposing-statement question format. Various items within the same construct group were randomized to avoid systemic response bias. PEOU and PU were measured by using six and four items in the survey, respectively³.

The following hypotheses were formulated:

- $H1_0$: FeDRE² is perceived as difficult to use,
 $H1_a$: FeDRE² is perceived as easy to use.
- $H2_0$: FeDRE² is perceived as not useful,
 $H2_a$: FeDRE² is perceived as useful.

In addition, it was also defined a correct solution for a requirements evolution. The aim was to compare the subjects' solutions with the corrected solution in order to analyze the degree in which the subjects applied FeDRE² in an effective and efficient way.

Thus, six objective dependent variables were defined:

- *Effectiveness_EVOSCEN*, which is calculated as the ratio between the number of right evolution scenarios that the subject identified and the total number of right evolution scenarios.
- *Effectiveness_EVO*, which is calculated as the ratio between the number of right requirements evolutions that the subject identified and the total number of right requirements evolutions.
- *Effectiveness_UPTRACE*, which is calculated as the ratio between the number of right updates in the traceability matrix that the subject performed and the total number of right updates in the traceability matrix.
- *Efficiency_EVOSCEN*, which is calculated as the ratio between the number of right evolution scenarios that the subject identified and the total time spent on the evolution scenarios identification.
- *Efficiency_EVO*, which is calculated as the ratio between the number of right requirements evolutions that the subject performed and the total time spent on the requirements evolutions.
- *Efficiency_UPTRACE*, which is calculated as the ratio between the number of right updates in the traceability matrix that the subject performed and the total time spent on the updates in the traceability matrix.

Table 6.3 shows the empirical study design. This design was used within both sessions (Brazil and Spain). It took us 3 days, with a total of 3 hours per day, to perform the empirical study for each session. In the first day, it was performed a 3 hour training session, which included

³The relationship between the survey statements with the dependent variables is presented in Appendix D.1

Table 6.3: Empirical Study Design.

Day	Topic	Activity
First	FeDRE Training (3 hours)	- Introduce FeDRE - Perform an Exercise
Second	FeDRE ² Training (3 hours)	- Introduce FeDRE ² - Perform an Exercise
Third	Collect Subject Background (15 min.)	-Apply the Background Questionnaire
	Perform the Empirical Study (2h30min.)	- Identify Evolution Scenarios - Perform SPL RE Evolution - Update the Traceability Matrix
	Collect Subject Feedback (15 min)	- Apply the Survey

the presentation of FeDRE approach to specify SPL requirements and a set of exercises to use FeDRE. The next day, it was presented FeDRE² to evolve SPL requirements and also executed some exercises to practice the approach. Finally, in the last day, we performed the empirical study. In the first 15 minutes we asked the subjects to answer a background form, and then, they were asked to perform the 3 tasks of FeDRE². After performing the 3 tasks, the subjects answered a survey about FeDRE².

6.4.2 Preparation of the Empirical Study

Several documents were designed as instrumentation for the empirical study: slides for the training session, an explanation of both methods, a data gathering form, and a questionnaire. These documents were used by the subjects, which were chosen for convenience from a group of software engineering research associates. The subjects were asked about their experience in the area, and the results showed that none of them had a previous background in this context. As a consequence, we did not establish a classification of subjects based on their experience in SPL requirements evolution. The first session was composed of 3 Bachelors and 6 Masters, totalizing 9 subjects from the Federal University of Bahia (UFBA, Brazil). The second session was composed of 6 Masters and 1 Ph.D, totalizing 7 subjects from Universitat Politècnica de València (UPV, Spain). Thus, the total of subjects for the empirical study was 16.

For training FeDRE and FeDRE² approaches, it was built an SPL in the car domain. With regard to the training of FeDRE approach, all the artifacts⁴ (*i.e.*, Feature Model, Feature Specification and Product Map) were created by a Ph.D. student. The feature model had 15

⁴FeDRE training artifacts are available at: <http://goo.gl/rMdXKi>

features in its specifications, the glossary had 5 terms, and the product map had 4 products. After the FeDRE training, the subjects were asked to specify the requirements for a sub-set of 3 features from the feature model. With regard to the training of FeDRE² approach, all the artifacts⁵ (*i.e.*, Change Request, Feature Model, Feature Specification, Glossary, Use Case Specification, and Traceability Matrix) were created by the Ph.D. student. The change request artifact had 2 tickets⁶, feature model had 15 features with its specifications, the glossary had 5 terms, the use case specification included 2 textual use case specifications plus its use case diagram specification, and the traceability matrix had the relationships among the 2 use cases and its features. After the FeDRE² training, the subjects were asked to evolve the SPL requirements according to the 2 change requests.

Before starting the empirical study, the subjects were asked to answer a background form⁷, which was elaborated with 16 questions including the subject's experience within SPL, requirements engineering, and software evolution.

The SAVi SPL was used for the FeDRE² empirical study. The artifacts⁸ (*i.e.*, Change Request, Feature Model, Feature Specification, Glossary, Use Case Specification, and Traceability Matrix) were created by Master/Ph.D. students of RiSE Labs⁹, one domain analyst and one domain expert, who were also assisted by a scoping expert with more than 6 years of experience in SPL scoping activities. Thus, the change request had 4 tickets, the feature model had 12 features with its specifications, the glossary had 16 terms, the use case specification included 19 textual use case specifications plus its use case diagrams specifications, and the traceability matrix had the relationships among the 19 use cases its features. It was performed an empirical study using features with a high level of granularity. For example, the *Access_Control* feature is responsible for *Creating an user*, *Performing the login*, *Showing the user Profile*, *Remembering the user password*, and finally *Sending a email with the new password to the user*.

After finishing the empirical study, the subjects were asked to answer a survey¹⁰, which was elaborated with 14 questions (open and closed) about the subject's opinion of the approach.

⁵FeDRE² training artifacts are available at: <http://goo.gl/xS1QDw>

⁶Each Ticket corresponds to a change request with an unique ID.

⁷The background form is shown in Appendix D.2

⁸FeDRE² empirical study artifacts are available at: <http://goo.gl/nRZC7J>

⁹RiSE Labs: <http://www.rise.com.br>

¹⁰The applied survey is presented in Appendix D.3

6.4.3 Data Collection

The SPL functional requirements were evolved by sixteen subjects (Bachelors, Masters, and Ph.D.) from both universities (Brazil and Spain). The data was collected before, during, and after the empirical study.

Before the empirical study, it was collected data from the subject's background. During the empirical study, the SPL functional requirements were evolved by applying the guidelines for evolving SPL functional requirements in order to deal with the following tasks: 1) *identify the evolution scenario (what)*; 2) *evolve the SPL requirements (how)*; and 3) *Update the traceability matrix*. After the empirical study, we were able to collect data from the subjects through a survey.

6.4.3.1 Background Form

It took us around 15 minutes, for each session (Brazil and Spain), to collect the information about the subject's experience on SPL, requirements engineering, and software evolution. Most of the subjects knew what SPL are and they have, in general, less than 1 year of experience within SPL. Moreover, the majority of the subjects knew what requirements engineering and software evolution are and have between 1 and 5 years of experience within them.

6.4.3.2 Empirical Study

It took around 2 hours and 30 minutes, for each session (Brazil and Spain), to collect the subject's answers about the evolution of the SPL requirements. The data was collected for each one of the following tasks:

- ***Identify the Evolution Scenario (what)***. All the subjects received 4 change requests to perform the SPL requirements evolution. These change requests were created by one of the authors of the approach. The first change request (*Update a photo for the user profile*) was a *Refine an Use Case* evolution scenario. The second change request (*Import Contacts from Facebook*) was an *Add New Alternative (XOR) Feature* evolution scenario. The third change request (*Notification on Facebook*) was an *Add New Alternative (OR) Feature* evolution scenario. Finally, the fourth change request (*Tracking the User*) was an *Add a New Optional Feature* evolution scenario. Based on the change requests, around 76% of the subjects identified the correct evolution scenarios.

- ***Evolve the SPL Requirements (how)***. Once identified the evolution scenarios, the subjects should evolve the SPL requirements, textual and diagram specifications. For the *Refine an*

Use Case evolution scenario, the subjects should evolve the *Update User Profile* textual use case specification by improving its steps within the main scenario. According to the *Add New Alternative (XOR) Feature* evolution scenario, the subjects should include a new XOR feature (*Facebook_Import*) in the feature model and evolve the *Retrieve Contacts* textual use case specification by adding a new alternative scenario associated with the new feature. For the *Add New Alternative (OR) Feature* evolution scenario, the subjects should include a new OR feature in the feature model (*Facebook_Destination*) and evolve the *Send Message* textual use case specification by adding a new alternative scenario associated with the new feature. Lastly, according to the *Add a New Optional Feature* evolution scenario, the subjects should create a new feature in the feature model (*Tracking User*) and specify a new textual use case (for example: *Track User*) associated with the new feature. Within this last evolution, it is also necessary to specify the use case diagram for the *Track User* textual specification. Based on the SPL requirements evolution, around 91% of the subjects evolved the SPL requirements correctly.

- **Update the Traceability Matrix.** Last but not least, the subjects should update the traceability matrix. They should include 3 new features and 1 use case within the traceability matrix. The first new feature (*Facebook_Import*) should be associated with the *Retrieve Contacts* use case. The second new feature (*Facebook_Destination*) should be associated with the *Send Message* use case. Finally, the third new feature (*Tracking User*) should be associated with the *Track User* use case. Based on the update within the traceability matrix, around 91% of the subjects updated it correctly.

6.4.3.3 Survey

To collect the survey information, for each session (Brazil and Spain), it took around 15 minutes. The open questions of the survey revealed that 12.5% of the subjects believe that the guidelines has lots of steps, which could be reduced, and 18.75% of the subjects suggested a tool support for the approach. Around 6.25% of the subjects also complained about too much natural language within the change requests, which may introduces ambiguity and sometimes it can make difficult the identification of the scenario to be evolved (*i.e.*, deciding if it is necessary to add new features or new uses cases). Thus, 6.25% of the subjects suggested an application for specifying these change requests in a rigorous way, allowing the automatic identification of the evolution scenario. Moreover, 6.25% of the subjects suggested that there is a need for supporting the evolution of features that have dependency (requires and excludes) with other features in the feature model; 6.25% complained about the feature description, which should be better explained; and

6.25% suggested that the approach should deal with the merge evolution of use cases. Besides the related difficulties, 37.5% of the subjects explicitly reported that the guidelines are well explained and easy to follow representing a systematic way to evolve SPL requirements.

6.4.4 Data Analysis

It was performed two quantitative analyses for the collected data. The first analysis was related to the objective variables (*effectiveness*, *efficiency*) observed during the execution of the tasks. This analysis was performed in order to identify deficiencies in the guidelines and improve the redaction in a qualitative manner. The second quantitative analysis was performed by using closed questions, which were filled in by the subjects after the empirical study execution. This information was analyzed in a quantitative manner in order to check the results of the subjects' *perception of ease of use* and *usefulness*, and their statistical relevance.

6.4.4.1 First Quantitative Analysis

The first quantitative analysis was performed based on the 3 tasks that the subjects performed within the empirical study. The results of the quantitative analysis is shown in Table 6.4. Regarding effectiveness in task 1 of the empirical study, we measured the quotient of the right number of evolution scenarios identified by the total number of evolution scenarios written in the solution (Effectiveness_EVOSCEN). The users were able to identify correctly 76.5% of the total evolution scenarios within task 1. In order to check the effectiveness in task 2 of the empirical study, we compared the evolved SPL requirements with the evolved SPL requirements written in the solution (Effectiveness_EVO). The results show that this variable has a mean of 0.9179, meaning that the users were able to correctly evolve 91.7% of the total requested evolutions. To check the effectiveness within task 3, we compared the updated traceability matrices with the updated traceability matrix written in the solution (Effectiveness_UPTRACE). The results show that this variable has a mean of 0.9166, meaning that the users were able to correctly update 91.6% of the total updates in the traceability matrix. One may complain about the total percentage of the corrected evolution scenarios (76.5%) is lower than the total percentage of corrected evolved SPL requirements (91.7%) and also lower than the total percentage of corrected updates in the traceability matrix (91.6%). These percentages should vary at same level since a correct evolution scenario would lead to a correct evolution within the SPL requirements and traceability matrix and vice versa. However, since our results showed that the percentage of correct evolution scenarios is lower than the others, we made a deep investigation within our data and realized that some subjects misunderstood two evolution scenarios (Add New Alternative

Table 6.4: Mean and Standard Deviation for the Analyzed Objective Dependent Variables.

Objective Dependent Variable	Mean	Standard Deviation (SD)
<i>Effectiveness_EVOSCEN</i>	0.7656	0.2953
<i>Efficiency_EVOSCEN</i>	0.2088	0.1591
<i>Time Task 1 (minutes)</i>	24.1875	20.4065
<i>Effectiveness_EVO</i>	0.9179	0.1223
<i>Efficiency_EVO</i>	0.2845	0.0919
<i>Time Task 2 (minutes)</i>	28.1875	8.7880
<i>Effectiveness_UPTRACE</i>	0.9166	0.2277
<i>Efficiency_UPTRACE</i>	1.1364	0.7955
<i>Time Task 3 (minutes)</i>	3.0625	1.3400

(XOR) Feature and Add New Alternative (OR) Feature). These two evolution scenarios have very similar steps for evolving the SPL requirements and updating the traceability matrix. Thus, even though the subjects have chosen the wrong evolution scenario (Add New Alternative (XOR) Feature or Add New Alternative (OR) Feature), they made the right evolutions within the SPL requirements and traceability matrix.

Additionally, for each task it was measured the time used and the efficiency estimation. The results show that the subjects took around 24 minutes to complete the task 1, with an *Efficiency_EVOSCEN* value of 0.2088 (number of right evolution scenarios / time). The subjects took around 28 minutes to complete the task 2, with an *Efficiency_EVO* value of 0.2845 (number of right SPL requirements evolutions / time). And finally, the results show that the subjects took around 3 minutes to complete the task 3, with an *Efficiency_UPTRACE* value of 1.1364 (number of right updates in the traceability matrix / time).

6.4.4.2 Second Quantitative Analysis

This Section discusses the results of the empirical study by quantitatively analyzing the data according to the stated hypotheses. All the results presented were obtained by using the SPSS v20 statistical tool with an alpha value of 0.05. The subjective variables were analyzed by comparing whether the mean of the responses¹¹ to the questions related to each variable were significantly greater than the Likert neutral value (equal to 3). In our case, the mean variable ranging from 1 to 5 for the measurement of both subjective variables has been considered as an interval scale (Carifio and Perla, 2007). Both variables have a mean over the neutral value 3 (see Table 6.5). In order to verify the hypotheses with this data, we first selected which test

¹¹Subjects responses are available in Appendix D.4

Table 6.5: Analysis of PEOU and PU Variables.

Subjective Dependent Variable	Mean	SD	Shapiro-Wilk	Cronbach's Alpha	Wilcoxon Test
PEOU	4.1250	0.9159	0.0065	0.9254	0.003
PU	4.0781	0.9251	0.0221	0.8234	0.002

was most appropriate for the data. It was first necessary to check whether the data was normally distributed. Since the sample size is smaller than 50, we applied the Shapiro-Wilk test (Shapiro and Wilk, 1965) to verify whether the data is normally distributed. The results of the normality test (Table 6.5) show that both variables are not normally distributed in this evaluation method since the results are smaller than 0.05. Thus, we check the hypotheses by performing a Wilcoxon Signed-Rank Test with a hypothesized median value of 3 (one sample nonparametric test). The p-values obtained for Wilcoxon Test (Table 6.5) were ≤ 0.05 ($p \leq \alpha$). As a consequence, we reject both null hypotheses; accepting that FeDRE² is perceived as easy to use and useful.

The box plots for the two subjective dependent variables (PEOU and PU) were also analyzed. Thus, for each box plot, it was analyzed the medians, the interquartile ranges (the box lengths), the overall spreads (distances between adjacent values), and the skewness to draw more conclusions.

Within the first box plot (Figure 6.9) it was analyzed the two variables in general for both empirical studies sessions (Brazil and Spain). Both medians are above the neutral value of 3 (PEOU = 4.3333 and PU = 4.5000). The length of the PU box is bigger than the PEOU box, thus, the range of the answers for PU variable vary more than PEOU. The overall spreads are similar considering the outlier (subject 13) from PEOU box plot. The box plot for PEOU shows a slight lower-skew: the lower whisker is longer than the upper one. The same happens for PU box plot where a lower-skew is observed: the lower whisker is longer than the upper. This shows that most of the answers for PEOU and PU are below their medians.

In the box plot from Figure 6.10, it was compared the PEOU subjective dependent variable between both empirical studies sessions (Brazil and Spain). Both medians are above the neutral value of 3 (PEOU_Brazil = 4.6666 and PEOU_Spain = 3.8333). The length of the PEOU_Spain box is slight bigger than the PEOU_Brazil box, thus, the range of the answers for PEOU_Spain variable vary more than PEOU_Brazil. Moreover, the overall spreads are different, the PEOU_Spain variable has a larger spread of the results. The box plot for PEOU_Brazil shows a lower-skew: the lower whisker is longer than the upper one. The same happens for PEOU_Spain box plot where a lower-skew is also observed: the lower whisker is longer than the upper. This fact shows that most of the answers for PEOU_Brazil and PEOU_Spain are

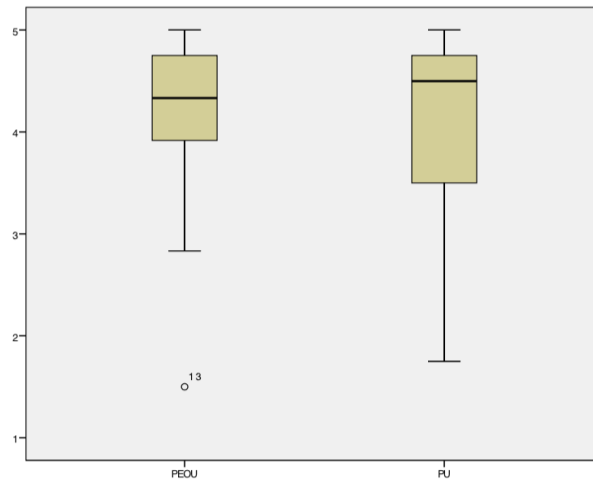


Figure 6.9: Perceived Ease of Use and Perceived Usefulness Box Plot.

below their medians. It is worth to notice that the outlier (subject 13) for PEOU subjective dependent variable is a subject from Spain and made some useful comments about the approach. According to this subject survey answers, this subject had some difficulties within identifying the SPL evolution scenario from task 1 and suggested to turn this task more rigorous, avoiding misunderstandings within the natural language.

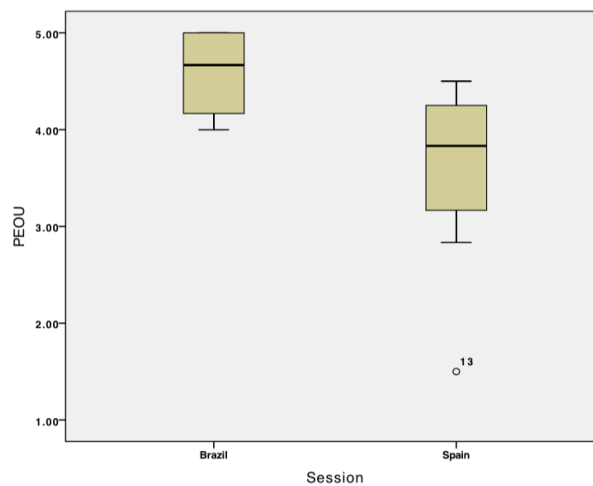


Figure 6.10: Perceived Ease of Use Box Plot, for each Session (Brazil and Spain).

Within the box plot from Figure 6.11, it was compared the PU subjective dependent variable between both empirical studies sessions (Brazil and Spain). Both medians are above the neutral value of 3 (PU_Brazil = 4.5000 and PU_Spain = 3.7500). The length of the PU_Spain box is bigger than the PU_Brazil box, thus, the range of the answers for PU_Spain variable vary more

than PU_Brazil. Moreover, the overall spreads are different, the PU_Spain variable has also a larger spread within the results. The box plot for PU_Brazil shows a slight lower-skew: the lower whisker is slight longer than the upper one. The same happens for PU_Spain box plot where a slight lower-skew is also observed: the lower whisker is slight longer than the upper one. Thus, the box plots show that most of the answers for PU_Brazil and PU_Spain are below their medians. Moreover, there is an outlier (subject 1) for PU subjective dependent variable within the Brazil Session. According to this subject survey answers, FeDRE² can increase the time for evolving SPL requirements.

We could realize, based on box plots (from Figures 6.10 and 6.11) that the Spanish group perceive FeDRE² less easy to use and useful than the Brazilian one. We believe that this happened because the empirical study was applied by another researcher in Spain, and in Brazil the empirical study was applied by one of the authors of the approach.

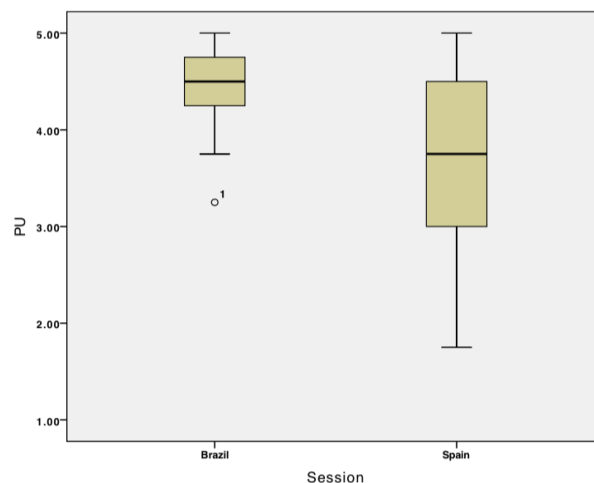


Figure 6.11: Perceived Usefulness Box Plot, for each Session (Brazil and Spain).

In order to check if there is a significant difference among the data from PEOU and PU variables for each Session (Brazil and Spain), we also test our data, which has a non-normal distribution, with the Mann-Whitney U Test (Table 6.6). The results of this statistical test shows if there is a significant difference between two groups of data. For PEOU variable (Figure 6.10), the Mann-Whitney U Test result was <0.05 , indicating a significant difference between the two sessions. This difference happened because of the Spanish outlier (subject 13). Within the PU variable (Figure 6.11), the Mann-Whitney U Test result was ≥ 0.05 , indicating that there was no significant difference between the two sessions.

6.4.5 Threats to Validity

The main threats to the **internal validity** of the empirical study are: empirical study design, evaluation design, subject experience, information exchange among evaluators, and the understandability of the documents. It was selected the empirical study design based on the methods proposed by [Abrahão *et al.* \(2011\)](#) and [Davis \(1989\)](#) (TAM) to evaluate FeDRE². There are some drawbacks related to TAM, as presented by [Turner *et al.* \(2008\)](#). However, since there is no approach with similar objectives such as the ones from FeDRE², it was used these methods ([Davis, 1989](#); [Abrahão *et al.*, 2011](#)) to evaluate FeDRE². It is important to highlight that these methods are still been used nowadays to evaluate new proposed approaches ([Oliveira *et al.*, 2013](#); [Fernandez *et al.*, 2013](#); [Molina *et al.*, 2013, 2014](#); [Asadi *et al.*, 2015](#)). The evaluation design might have affected the results owing to the creation of the change requests taken as input to evolve the SPL requirements when applying FeDRE². We attempted to alleviate this threat by considering change requests, which implied applying most of the FeDRE² guidelines rules. Subject experience was mitigated owing to the fact that none of the subjects had any experience in SPL requirements evolution. Moreover, we assume that using students as subjects do not affect the overall results of software engineering experiments, since the perform of students and practitioners are similar when applying a new approach ([Salman *et al.*, 2015](#)). Information exchange was mitigated by monitoring the participants while they performed the tasks. The empirical study was performed in two sessions but no relationships were established between Brazilian and Spanish subjects, and no information was exchanged among them. The understandability of the material was alleviated by clearing up all the misunderstandings that appeared in each session.

The main threat to the **external validity** is the generalizability of the results. To allviate this threat, the empirical study was performed in Brazil and replicated in Spain. We had different results from both countries, were the Brazilian group perceived FeDRE² as more easy to use and useful than the Spanish group. Although the results have shown FeDRE² as a promising approach, it needs a broader evaluation in order to try to generalize the results.

The main threat to the **construct validity** of the empirical study was the reliability of the questionnaire, related to the two empirical study hypotheses. This reliability was tested by

Table 6.6: Mann-Whitney U Test Analysis of PEOU and PU Variables for Each Session.

Subjective Dependent Variable	Mann-Whitney Test
Perceived Ease of Use (PEOU)	0.0104
Perceived Usefulness (PU)	0.1334

applying the Cronbach's alpha test (Table 6.5) to each set of closed questions which measured the PEOU and PU variables, obtaining a value of 0.9254, and 0.8234 respectively (higher than the minimum acceptance threshold $\alpha=0.70$) (Maxwell, 2002). Moreover, there is a threat related to the representativeness of the results. To alleviate this threat and make the tasks enough representative, the complexity of the exercise was adjusted for the subjects to be able to apply most of the guidelines rules, considering that the duration of the experiment was limited to 150 minutes.

The main threat to the **conclusion validity** of the experiment was the pattern recognition (recall and precision) for the effectiveness and efficiency of FeDRE² activities, and the validity of the statistical test applied. For this empirical study we used only the recall pattern to evaluate the effectiveness and efficiency of FeDRE² activities, however, this is a well know and used pattern. The validity of the statistical test applied was alleviated by applying the most common test that is employed in the empirical software engineering field (Maxwell, 2002).

6.5 Chapter Summary

This Chapter introduced the FeDRE² approach to support the evolution of SPL requirements. Within this approach, the evolution of SPL requirements is performed in a systematic way through a set of guidelines. The guidelines are composed by three main tasks (1. Identify the Evolution Scenario, 2. Perform the Evolution, and 3. Update the Traceability Matrix), which have several steps to be executed in order to evolve the SPL requirements. Thus, through the proposed approach, the SPL requirements can be evolved in a safe way keeping its traceability within other artifacts. We believe that this approach provides a solution to make easier and achieve the intended SPL RE evolution.

The feasibility of FeDRE² was evaluated using an empirical study involving a mobile application for emergency notifications. The results show that the subjects perceived the approach as easy to use and useful for evolving the functional requirements in this particular SPL. However, the approach needs further empirical evaluation with larger and more complex SPL. Moreover, some steps may be taken to improve the approach, such as: reducing the number of steps within the guidelines; building a tool support, which was also a need found by Alves *et al.* (2010); reducing the natural language within the identification of evolution scenarios, which will require a more rigorous process; supporting the evolution of features with dependencies (requires and excludes) in the feature model; and dealing with the merge evolution of use cases. Next Chapter presents the conclusions and future work for this Thesis.

PART V

Conclusions and Future Work

7

Conclusions

As stated in previous work ([Lehman \(1996\)](#); [Lehman *et al.* \(1997\)](#); Chapter 3; and Chapter 4), the system must support changes, otherwise it will gradually lapse into uselessness.

These changes in Software Product Lines (SPL) are firstly represented by requirements. Thus, SPL needs to provide mechanisms to deal with the changes that may occur within the requirements. However, so far, there is a lack of approaches dealing in a systematic way within the SPL requirements specification and evolution, as stated by [Alves *et al.* \(2010\)](#) and [Oliveira *et al.* \(2015d\)](#). Thus, the contribution of this Thesis was to understand how SPL evolves during the time and to propose two approaches to deal with the SPL requirements specification and evolution, respectively.

Part III of the Thesis (Chapter 3) improved the SPL evolution knowledge by performing three studies to understand the SPL evolution: two studies evaluating the Lehman's Laws of software evolution within two industrial SPL projects; and one systematic mapping study on SPL evolution, describing approaches that deal with SPL evolution and gaps for future research.

Latter, in Part IV of the Thesis, it was proposed two approaches to deal with SPL requirements specification and evolution, FeDRE and FeDRE² approaches respectively (Chapter 5 and Chapter 6). Both approaches were empirically evaluated and the results shown that, besides being a first evaluation, they were perceived as easy to use and useful by the subjects from Brazil and Spain.

The work performed here was a first step within improving the SPL evolution, since it is based on requirements engineering activities. Based on the performed studies and on the proposed approaches there are some gaps identified for future research as shown on the next Section 7.1. Next, Section 7.2 discusses the main related work. Finally, Section 7.3 presents the main contribution of this Thesis.

7.1 Future Work

The investigation performed so far revealed some gaps for future research. They can be summarized as shown next.

7.1.1 *Evaluating Lehman's Laws (LL) of Software Evolution*

- Measurements within the SPL common, variable and product-specific assets (including requirements, architecture, code, and so on) should be part of the SPL evolution process. Thus, after evolving the SPL assets, measurements may be applied to check if the new change increased or not the complexity and quality of the SPL.
- The management team should be aware of the clients' needs in order to keep adding/removing functionalities within the SPL. However, this should be performed in a systematic way, paying attention to not increase LOCs of the assets, which may lead to increase in complexity.
- More studies evaluating the LL within industrial SPL projects should be performed to strengthen the results obtained so far. Thus, the guideline for evolving SPL assets should be updated based on the results of the new studies. For example, the guideline should incorporate steps for dealing with the declining quality and increase complexity laws during the SPL evolution, if these laws were confirmed within the new studies.
- Within the laws not supported in SPLs, it is important to understand the reason why and maybe propose new laws of evolution for SPLs.

7.1.2 *Systematic Mapping Study on SPL Evolution*

- Since this study is based on a taxonomy for software change ([Buckley et al., 2005](#)), a future work may be the proposing of a taxonomy for SPL evolution. This taxonomy should take into account where most of the studies fitted according to the data extraction form and try to identify what is specific to SPL.
- This study revealed several gaps for future research (as shown in [Table 4.11](#)). One of findings is related to the lack of approaches dealing with the evolution of requirements within SPL application engineering. This finding is complementary to the work performed in this Thesis. Since the investigation performed in this Thesis focused on the SPL domain engineering, it is also important to propose an approach to deal with the evolution of SPL products requirements for the SPL application engineering. Moreover, another

finding of the systematic mapping study was to perform evaluations through case studies in the industry. Thus, such approach for evolving SPL requirements in the application engineering may be evaluated through an industry case study.

7.1.3 *Feature-Driven Requirements Engineering (FeDRE) Approach*

- The approach needs further empirical evaluation with larger and more complex SPLs.
- It is interesting that the guidelines proposed by FeDRE incorporates how to deal with feature dependencies (requires and excludes). Moreover, the guidelines should deal with quality attributes (*i.e.*, the complexity of an use case and the complexity of an use case diagram, as stated by [Montagud et al. \(2012\)](#)).
- There is a need for building a tool to support the specification proposed by FeDRE and also its guidelines, which was a need also found by [Alves et al. \(2010\)](#).

7.1.4 *Feature-Driven Requirements Engineering Evolution (FeDRE²) Approach*

- The approach needs further empirical evaluation with larger and more complex SPLs.
- Moreover, the approach may be improved by following these suggestions: reducing the number of steps within the guidelines; reducing the natural language within the identification of evolution scenarios, which will require a more rigorous process; supporting the evolution of features with dependencies (requires and excludes) in the feature model; and dealing with the merge evolution of use cases.
- There is a need to deal with the evolution within the next phases of the SPL life cycle (*i.e.*, architecture, code, tests). Thus, a next step should be the extension of the approach to support the evolution of the whole artifacts SPL (mainly, how they co-evolve over time).
- There is a need for building a tool to support the evolution guidelines proposed by FeDRE².

7.2 Related Work

In literature, there are several proposals to improve the SPL evolution. We systematically analyzed the literature on SPL evolution, as earlier addressed in Chapter 4. Moreover, the most important publications in the SPL field, related to SPL requirements specification and evolution, have been discussed in this Thesis (Chapter 5 and Chapter 6). However, the key difference between the investigation of this Thesis and others work is the attempt to systematize

the SPL requirements specification and evolution through the use of guidelines and the empirical evaluation of the proposed approaches within two different countries (Brazil and Spain).

7.3 Main Contributions

This Thesis described earlier, in the Section 1.5, the main contributions expected from this investigation. Some of the results have been already published. Others were submitted to relevant conferences and journals of the field. Next, it is listed the set of papers resulting from this investigation.

Published:

- Oliveira, R. P., Insfran, E., Abrahao, S., Gonzalez-Huerta, J., Blanes, D., Cohen, S., and de Almeida, E. S. (2013). *A feature-driven requirements engineering approach for software product lines*. In VII Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS), pages 1–10.
- Oliveira, R. P., Blanes, D., Gonzalez-Huerta, J., Insfran, E., Abrahao, S., Cohen, S., and Almeida, E. S. (2014). *Defining and validating a feature-driven requirements engineering approach*. Journal of Universal Computer Science (JUCS), 20(5), 666–691.
- Oliveira, R. P., Almeida, E. S., and Gomes, G. S. S. (2015). *Evaluating lehman’s laws of software evolution within software product lines: A preliminary empirical study*. In Proceedings of the 14th International Conference on Software Reuse (ICSR), pages 42–57.
- Oliveira, R. P. and Almeida, E. S. (2015). *Requirements evolution in software product lines: An empirical study*. Submitted to Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS), pages 1–10.

Submitted:

- Oliveira, R. P., Santos, A. R., Almeida, E. S., and Gomes, G. S. S. (2015). *Lehman’s laws of software evolution and software product lines: Empirical studies*. Submitted to Journal of Systems and Software (JSS), ICSR Special Issue.
- Oliveira, R. P., Santos, A., Almeida, E. S., Abrahao, S., and Insfran, E. (2015). *Software product lines evolution: A systematic mapping study*. Submitted to ACM Computing Surveys Journal.
- Oliveira, R. P., Santos, A. R., Almeida, E. S., and Gomes, G. S. S. (2015). *Lehman’s laws of software evolution and software product lines*. Submitted to IEEE Software Journal.

- Oliveira, R. P. and Almeida, E. S. (2015). *Guiding software product line evolution based on requirements engineering activities*. Submitted to Information and Software Technology (IST) Journal.

Finally, it is presented other important publications, not directly related within this Thesis, but still relevant to the SPL field.

Published:

- Souza, I. S., de Oliveira, R. P., da Silva Gomes, G. S., and de Almeida, E. S. (2012). *On the relationship between inspection and evolution in software product lines: An exploratory study*. In Brazilian Symposium on Software Engineering (SBES), pages 131–140.
- Souza, I. S., Fiaccone, R., de Oliveira, R. P., and de Almeida, E. S. (2013). *On the relationship between features granularity and non-conformities in software product lines: An exploratory study*. In Brazilian Symposium on Software Engineering (SBES), pages 147–156.
- Santos, A. R., de Oliveira, R. P., and de Almeida, E. S. (2015). *Strategies for consistency checking on software product lines: A mapping study*. In Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering (EASE), pages 1–14.

References

- Abrahão, S., Insfran, E., Carsí, J. A., and Genero, M. (2011). Evaluating requirements modeling methods based on user perceptions: A family of experiments. *Information Sciences*, **181**(16), 3356–3378.
- Ajila, S. and Kaba, A. (2004). Using traceability mechanisms to support software product line evolution. In *Proceedings of the IEEE International Conference on Information Reuse and Integration (IRI)*, pages 157–162.
- Alfárez, M., Lopez-Herrejon, R. E., Moreira, A., Amaral, V., and Egyed, A. (2011). Supporting consistency checking between features and software product line use scenarios. In *Proceedings of the 12th International Conference on Software Reuse (ICSR)*, pages 20–35.
- Alves, V., Gheyi, R., Massoni, T., Kulesza, U., Borba, P., and Lucena, C. (2006). Refactoring product lines. In *Proceedings of the 5th International Conference on Generative Programming and Component Engineering (GPCE)*, pages 201–210.
- Alves, V., Niu, N., Alves, C., and Valença, G. (2010). Requirements engineering for software product lines: A systematic literature review. *Information and Software Technology (IST)*, **52**(8), 806–820.
- Anquetil, N., Kulesza, U., Mitschke, R., Moreira, A., Royer, J.-C., Rummler, A., and Sousa, A. (2010). A model-driven traceability framework for software product lines. *Software Systems Modeling*, **9**(4), 427–451.
- Asadi, M., Bagheri, E., Gašević, D., Hatala, M., and Mohabbati, B. (2011). Goal-driven software product line engineering. In *Proceedings of the 2011 ACM Symposium on Applied Computing (SAC)*, pages 691–698.
- Asadi, M., Soltani, S., Gasevic, D., and Hatala, M. (2015). The effects of visualization and interaction techniques on feature model configuration. *Empirical Software Engineering*, pages 1–38.
- Assunção, W. K. G. and Vergilio, S. R. (2014). Feature location for software product line migration: A mapping study. In *Proceedings of the 18th International Software Product Line Conference (SPLC)*, pages 52–59.

-
- Atkinson, C., Bayer, J., Bunse, C., Kamsties, E., Laitenberger, O., Laqua, R., Muthig, D., Paech, B., Wust, J., and Zettel, J. (2002). *Component-based Product Line Engineering with UML*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.
- Bailetti, A., Ajila, S., and Dumitrescu, R. (2004). Experience report on the effect of market reposition on product line evolution. In *Proceedings of the IEEE International Conference on Information Reuse and Integration (IRI)*, pages 151–156.
- Barry, E. J., Kemerer, C. F., and Slaughter, S. A. (2007). How software process automation affects software evolution: a longitudinal empirical analysis: Research articles. *Journal of Software Maintenance and Evolution: Research and Practice*, **19**, 1–31.
- Basili, V. R., Caldiera, G., and Rombach, H. D. (1994). *Goal Question Metric Paradigm*, pages 528–532. Wiley-Interscience.
- Bayer, J., Muthig, D., and Widen, T. (1999a). Customizable domain analysis. In *Generative and Component-Based Software Engineering (GCSE)*, pages 178–194.
- Bayer, J., Flege, O., Knauber, P., Laqua, R., Muthig, D., Schmid, K., Widen, T., and DeBaud, J.-M. (1999b). Pulse: A methodology to develop software product lines. In *Proceedings of the 1999 Symposium on Software Reusability*, pages 122–131.
- Birk, A., Heller, G., John, I., Joos, S., Muller, K., Schmid, K., and Massen, T. (2003). Report of the gi work group "requirements engineering for product lines". technical report.
- Bohner, S. and Arnold, R. (1996). *Software Change Impact Analysis*. IEEE Computer Society Press.
- Bonifácio, R. and Borba, P. (2009). Modeling scenario variability as crosscutting mechanisms. In *Proceedings of the 8th ACM International Conference on Aspect-oriented Software Development (AOSD)*, pages 125–136.
- Borba, P., Teixeira, L., and Gheyi, R. (2012). A theory of software product line refinement. *Theoretical Computer Science*, **455**(0), 2–30.
- Bosch, J. (2000). *Design and use of software architectures - adopting and evolving a product-line approach*. Addison-Wesley.
- Bosch, J. and Ran, A. (2000). Evolution of software product families. In *International Workshop Software Architectures for Product Families (IW-SAPF)*, pages 168–183.

- Botterweck, G. and Pleuss, A. (2014). Evolution of software product lines. In *Evolving Software Systems*, pages 265–295. Springer Berlin Heidelberg.
- Botterweck, G., Pleuss, A., Dhungana, D., Polzer, A., and Kowalewski, S. (2010). Evofm: Feature-driven planning of product-line evolution. In *Proceedings of the Workshop on Product Line Approaches in Software Engineering (PLEASE)*, pages 24–31.
- Buckley, J., Mens, T., Zenger, M., Rashid, A., and Kniesel, G. (2005). Towards a taxonomy of software change: Research articles. *Journal of Software Maintenance and Evolution: Research and Practice*, **17**(5), 309–332.
- Budgen, D., Turner, M., Brereton, P., and Kitchenham, B. (2008). Using mapping studies in software engineering. In *PPIG 2008: In 20th Annual Meeting of the Psychology of Programming Interest Group*, pages 195–204. Lancaster University.
- Carifio, J. and Perla, R. J. (2007). Ten common misunderstandings, misconceptions, persistent myths and urban legends about likert scales and likert response formats and their antidotes. *Journal of Social Sciences*, **3**(3), 106–116.
- Carver, J. (2010). Towards reporting guidelines for experimental replications: A proposal. In *Proceedings of the 1st International Workshop on Replication in Empirical Software Engineering Research*, International Conference on Software Engineering (ICSE).
- Chastek, G., Donohoe, P., Kang, K. C., and Thiel, S. (2001). Product line analysis: A practical introduction. technical report cmu/sei-2001-tr-001.
- Cheng, B. H. C. and Atlee, J. M. (2007). Research directions in requirements engineering. In *2007 Future of Software Engineering*, Future of Software Engineering (FOSE), pages 285–303.
- Clements, P. C. and Northrop, L. (2002). *Software Product Lines: Practices and Patterns*. SEI Series in Software Engineering. Addison-Wesley.
- Czarnecki, K. and Eisenecker, U. W. (2000). *Generative Programming: Methods, Tools, and Applications*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA.
- Davis, F. D. (1989). Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS Quarterly*, **13**(3), 319–340.
-

-
- Deelstra, S., Sinnema, M., Nijhuis, J., and Bosch, J. (2004). Cosvam: a technique for assessing software variability in software product families. In *IEEE International Conference on Software Maintenance (ICSM)*, pages 458–462.
- Eriksson, M., Börstler, J., and Borg, K. (2005). The pluss approach - domain modeling with features, use cases and use case realizations. pages 33–44.
- Fenske, W., Thüm, T., and Saake, G. (2014). A taxonomy of software product line reengineering. In *Proceedings of the Eighth International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS)*, pages 1–8.
- Fernandez, A., Insfran, E., and Abrahão, S. (2011). Usability evaluation methods for the web: A systematic mapping study. *Information and Software Technology (IST)*, **53**(8), 789–817.
- Fernandez, A., ao, S. A., and Insfran, E. (2013). Empirical validation of a usability inspection method for model-driven web development. *Journal of Systems and Soft. (JSS)*, **86**(1), 161–186.
- Fricker, S. and Stoiber, R. (2008). Relating product line context to requirements engineering processes using design rationale. In *Software Engineering (Workshops)*, pages 240–251.
- Godfrey, M. W. and German, D. M. (2014). On the evolution of lehman’s laws. *Journal of Software: Evolution and Process*, **26**(7), 613–619.
- Godfrey, M. W. and Tu, Q. (2000). Evolution in open source software: A case study. In *IEEE International Conference on Software Maintenance (ICSM)*, pages 131–142.
- Gomaa, H. (2013). Evolving software requirements and architectures using software product line concepts. In *International Workshop on the Twin Peaks of Req. and Arch. (TwinPeaks)*, pages 24–28.
- Gomaa, H. and Shin, M. (2008). Multiple-view modelling and meta-modelling of software product lines. *IET Software*, **2**, 94–122.
- Gonzalez-Barahona, J. M., Robles, G., Herraiz, I., and Ortega, F. (2014). Studying the laws of software evolution in a long-lived floss project. *Journal of Software: Evolution and Process*, **26**(7), 589–612.

- Griss, M. L., Favaro, J., and Alessandro, M. d. (1998). Integrating feature modeling with the reuse. In *Proceedings of the 5th International Conference on Software Reuse (ICSR)*, pages 76–85.
- Gupta, A., Cruzes, D., Shull, F., Conradi, R., Rønneberg, H., and Landre, E. (2010). An examination of change profiles in reusable and non-reusable software systems. *Journal of Software Maintenance and Evolution: Research and Practice*, **22**, 359–380.
- Heidenreich, F., Sánchez, P., Santos, J. a., Zschaler, S., Alférez, M., Araújo, J. a., Fuentes, L., Kulesza, U., Moreira, A., and Rashid, A. (2010). Transactions on aspect-oriented software development vii. chapter Relating Feature Models to Other Models of a Software Product Line: A Comparative Study of Featuremapper and VML, pages 69–114.
- Herraiz, I., Rodriguez, D., Robles, G., and Gonzalez-Barahona, J. M. (2013). The evolution of the laws of software evolution: A discussion based on a systematic literature review. *ACM Computing Surveys*, **46**(2), 1–28.
- Hesse-Biber, S. N. (2010). *Mixed methods research: merging theory with practice*. The Guilford Press, New York, NY, USA.
- Israeli, A. and Feitelson, D. G. (2010). The linux kernel as a case study in software evolution. *Journal of Systems and Software (JSS)*, **83**, 485–501.
- Jedlitschka, A., Ciolkowski, M., and Pfahl, D. (2008). Reporting experiments in software engineering. In F. Shull, J. Singer, and D. I. K. Sjøberg, editors, *Guide to Advanced Empirical Software Engineering*, pages 201–228. Springer London.
- John, I. and Eisenbarth, M. (2009). A decade of scoping: A survey. In *Proceedings of the 13th International Software Product Line Conference (SPLC)*, pages 31–40.
- Jones, C. (1991). *Applied Software Measurement: Assuring Productivity and Quality*. McGraw-Hill, Inc.
- Kan, S. H. (2002). *Metrics and Models in Software Quality Engineering*. Addison-Wesley, 2nd edition.
- Kang, K., Cohen, S., Hess, J., Nowak, W., and Peterson, S. (1990). *Feature-Oriented Domain Analysis (FODA) Feasibility Study*.
-

-
- Kemerer, C. and Slaughter, S. (1999). An empirical approach to studying software evolution. *IEEE Transactions on Software Engineering (TSE)*, **25**(4), 493–509.
- Kitchenham, B. and Charters, S. (2007). Guidelines for performing systematic literature reviews in software engineering. Technical Report EBSE 2007-001.
- Krueger, C. (2002a). Variation management for software production lines. In G. Chastek, editor, *Software Product Lines*, volume 2379 of *Lecture Notes in Computer Science*, pages 37–48. Springer Berlin Heidelberg.
- Krueger, C. W. (2002b). Easing the transition to software mass customization. In *Revised Papers from the 4th International Workshop on Software Product-Family Engineering (VaMoS)*, pages 282–293.
- Kwiatkowski, D., Phillips, P. C. B., Schmidt, P., and Shin, Y. (1992). Testing the null hypothesis of stationarity against the alternative of a unit root : How sure are we that economic time series have a unit root? *Journal of Econometrics*, **54**(1-3), 159–178.
- Laguna, M. A. and Crespo, Y. (2013). A systematic mapping study on software product line evolution: From legacy system reengineering to product line refactoring. *Science of Computer Programming*, **78**(8), 1010–1034.
- Lehman, M. (1980). Programs, life cycles, and laws of software evolution. *Proceedings of the IEEE*, **68**(9), 1060–1076.
- Lehman, M. M. (1996). Laws of software evolution revisited. In *Proceedings of the 5th European Workshop on Software Process Technology (EWSPT)*, pages 108–124.
- Lehman, M. M., Ramil, J. F., Wernick, P. D., Perry, D. E., and Turski, W. M. (1997). Metrics and laws of software evolution - the nineties view. In *Proceedings of the 4th International Symposium on Software Metrics*, pages 20–33.
- Lientz, B. P. and Swanson, B. E. (1980). *Software Maintenance Management: A Study of the Maintenance of Computer Application Software in 487 Data Processing Organizations*. Addison-Wesley.
- Lotufo, R., She, S., Berger, T., Czarnecki, K., and Wkasowski, A. (2010). Evolution of the linux kernel variability model. In *International Conference on Software Product Lines (SPLC)*, pages 136–150.

- Maxwell, K. (2002). *Applied Statistics for Software Managers*. Software Quality Institute Series, Prentice Hall.
- McCabe, T. J. (1976). A complexity measure. *IEEE Transactions on Software Engineering (TSE)*, **2**(4), 308–320.
- McGregor, J. D. (2003). The Evolution of Product Line Assets. Technical report.
- Mende, T., Beckwermert, F., Koschke, R., and Meier, G. (2008). Supporting the grow-and-prune model in software product lines evolution using clone detection. In *European Conference on Software Maintenance and Reengineering (CSMR)*, pages 163–172.
- Mens, T. and Demeyer, S. (2008). *Software Evolution*. Springer.
- Molina, A. I., Gallardo, J., Redondo, M. A., Ortega, M., and Giraldo, W. J. (2013). Metamodel-driven definition of a visual modeling language for specifying interactive groupware applications: An empirical study. *Journal of Systems and Soft. (JSS)*, **86**(7), 1772–1789.
- Molina, A. I., Redondo, M. A., Ortega, M., and Lacave, C. (2014). Evaluating a graphical notation for modeling collaborative learning activities: A family of experiments. *Science of Computer Programming*, **88**(0), 54–81.
- Montagud, S., Abrahão, S., and Insfrán, E. (2012). A systematic review of quality attributes and measures for software product lines. *Software Quality Journal*, **20**(3-4), 425–486.
- Moon, M., Yeom, K., and Chae, H. S. (2005). An approach to developing domain requirements as a core asset based on commonality and variability analysis in a product line. *IEEE Transactions on Software Engineering*, **31**(7), 551–569.
- Mussbacher, G., Araújo, J. a., Moreira, A., and Amyot, D. (2012). Aourn-based modeling and analysis of software product lines. *Software Quality Control*, **20**(3-4), 645–687.
- Naur, P. and Randell, B., editors (1969). *Software Engineering: Report of a Conference Sponsored by the NATO Science Committee, Garmisch, Germany, 7-11 Oct. 1968, Brussels, Scientific Affairs Division, NATO*.
- Neves, L., Teixeira, L., Sena, D., Alves, V., Kulezsa, U., and Borba, P. (2011). Investigating the safe evolution of software product lines. In *Proceedings of the 10th ACM international conference on Generative programming and component engineering (GPCE)*, pages 33–42.
-

-
- Northrop, L. M. (2002). Sei's software product line tenets. *IEEE Software*, **19**(4), 32–40.
- Oliveira, R. P. and Almeida, E. S. (2015a). Guiding software product line evolution based on requirements engineering activities. In *the Review Process. Submitted to the Information and Software Technology (IST) Journal*.
- Oliveira, R. P. and Almeida, E. S. (2015b). Requirements evolution in software product lines: An empirical study. In *Accepted at Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS)*, pages 1–10.
- Oliveira, R. P., Insfran, E., Abrahao, S., Gonzalez-Huerta, J., Blanes, D., Cohen, S., and de Almeida, E. S. (2013). A feature-driven requirements engineering approach for software product lines. In *VII Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS)*, pages 1–10.
- Oliveira, R. P., Blanes, D., Gonzalez-Huerta, J., Insfrán, E., Abrahão, S., Cohen, S., and Almeida, E. S. (2014). Defining and validating a feature-driven requirements engineering approach. *Journal of Universal Computer Science (JUCS)*, **20**(5), 666–691.
- Oliveira, R. P., Almeida, E. S., and Gomes, G. S. S. (2015a). Evaluating lehman's laws of software evolution within software product lines: A preliminary empirical study. In *Proceedings of the 14th International Conference on Software Reuse (ICSR)*, pages 42–57.
- Oliveira, R. P., Santos, A. R., Almeida, E. S., and Gomes, G. S. S. (2015b). Evaluating lehman's laws of software evolution within software product lines industrial projects. In *the Review Process. Submitted to Journal of Systems and Software (JSS), ICSR Special Issue*.
- Oliveira, R. P., Santos, A. R., Almeida, E. S., and Gomes, G. S. S. (2015c). Lehman's laws of software evolution and software product lines: Empirical studies. In *the Review Process. Submitted to IEEE Software*.
- Oliveira, R. P., Santos, A., Almeida, E. S., Abrahao, S., and Insfran, E. (2015d). Software product lines evolution: A systematic mapping study. In *the Review Process. Submitted to ACM Computing Surveys*.
- Petersen, K., Feldt, R., Mujtaba, S., and Mattsson, M. (2008). Systematic mapping studies in software engineering. In *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering (EASE)*, pages 68–77.

- Pohl, K., Böckle, G., and Linden, F. J. v. d. (2005). *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag New York, Inc.
- Pussinen, M. (2002). *A Survey on Software Product-line Evolution*. Tampere University of Technology.
- Ramil, J. F. and Lehman, M. M. (2000). Metrics of software evolution as effort predictors - a case study. In *IEEE International Conference on Software Maintenance (ICSM)*, pages 163–172.
- Runeson, P. and Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, **14**(2), 131–164.
- Salman, I., Misirli, A. T., and Juristo, N. (2015). Are students representatives of professionals in software engineering experiments? In *Proceedings of the International Conference on Software Engineering (ICSE)*, to appear.
- Santos, A. R., de Oliveira, R. P., and de Almeida, E. S. (2015a). Strategies for consistency checking on software product lines: A mapping study. In *Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering (EASE)*, pages 1–14.
- Santos, J. A., Santos, A. R., and de Mendonca, M. (2015b). Investigating bias in the search phase of software engineering secondary studies. In *Proceedings of the 12th Workshop on Experimental Software Engineering (ESELAW)* - to appear.
- Schulze, S., Thüm, T., Kuhlemann, M., and Saake, G. (2012). Variant-preserving refactoring in feature-oriented software product lines. In *Workshop on Variability Modeling of Software-Intensive Systems (VaMoS)*, pages 73–81.
- Shaker, P., Atlee, J. M., and Wang, S. (2012). A feature-oriented requirements modelling language. In *20th IEEE International Requirements Engineering Conference (RE)*, pages 151–160.
- Shapiro, S. S. and Wilk, M. B. (1965). An analysis of variance test for normality (complete samples). *Biometrika*, pages 591–611.
- Souza, I. S., de Oliveira, R. P., da Silva Gomes, G. S., and de Almeida, E. S. (2012). On the relationship between inspection and evolution in software product lines: An exploratory study. In *Brazilian Symposium on Software Engineering (SBES)*, pages 131–140.
-

-
- Souza, I. S., Fiaccone, R., de Oliveira, R. P., and de Almeida, E. S. (2013). On the relationship between features granularity and non-conformities in software product lines: An exploratory study. In *Brazilian Symposium on Software Engineering (SBES)*, pages 147–156.
- Svahnberg, M. and Bosch, J. (1999). Evolution in software product lines: two cases. *Journal of Software Maintenance and Evolution: Research and Practice*, **11**(6), 391–422.
- Thüm, T., Batory, D., and Kastner, C. (2009). Reasoning about edits to feature models. In *IEEE 31st International Conference on Software Engineering (ICSE)*, pages 254–264.
- Thurimella, A. and Bruegge, B. (2007). Evolution in product line requirements engineering: A rationale management approach. In *Requirements Engineering Conference, 2007. RE 07. 15th IEEE International*, pages 254–257.
- Turner, M., Kitchenham, B., Budgen, D., and Brereton, P. (2008). Lessons learnt undertaking a large-scale systematic literature review. In *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering (EASE)*, pages 110–118.
- Weiss, D. M. and Lai, C. T. R. (1999). *Software Product-line Engineering: A Family-based Software Development Process*. Addison-Wesley Longman Publishing Co., Inc.
- White, J., Galindo, J. A., Saxena, T., Dougherty, B., Benavides, D., and Schmidt, D. C. (2014). Evolving feature model configurations in software product lines. *Journal of Systems and Software (JSS)*, **87**, 119–136.
- Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., and Regnell, B. (2012). *Experimentation in Software Engineering*. Springer.
- Xie, G., Chen, J., and Neamtiu, I. (2009). Towards a better understanding of software evolution: An empirical study on open source software. In *IEEE International Conference on Software Maintenance (ICSM)*, pages 51–60.
- Yan, X. and Su, X. G. (2009). *Linear Regression Analysis: Theory and Computing*. World Scientific Publishing.
- Ye, H. and Liu, H. (2005). Approach to modelling feature variability and dependencies in software product lines. *IEE Proceedings - Software*, **152**(3), 101–109.
- Young, M. and Pezze, M. (2005). *Software Testing and Analysis: Process, Principles and Techniques*. John Wiley & Sons.

Appendices

A

Empirical Studies

This appendix presents some relevant data from the empirical studies performed to evaluate the applicability of Lehman's laws of software evolution in private SPLs, earlier discussed in [Chapter 3](#).

A.1 The KPSS Test and Hypotheses results (at MC)

Variable	Commonalities			Variabilities			Product-Specific		
	KPSS Test	p-value	Decision	KPSS Test	p-value	Decision	KPSS Test	p-value	Decision
NA	0.1651	0.0341	Reject H_0	0.2187	0.0100	Reject H_0	0.1979	0.0168	Reject H_0
LOC	0.2125	0.0113	Reject H_0	0.2400	0.0100	Reject H_0	0.1354	0.0697	Do Not Reject H_0
NCLOC	0.1856	0.0214	Reject H_0	0.2252	0.0100	Reject H_0	0.1764	0.0249	Reject H_0
NCM	0.1856	0.0214	Reject H_0	0.2274	0.0100	Reject H_0	0.1799	0.0236	Reject H_0
RGM	0.0508	0.1000	Do Not Reject H_0	0.0725	0.1000	Do Not Reject H_0	0.0491	0.1000	Do Not Reject H_0
NAD	0.0927	0.1000	Do Not Reject H_0	0.0911	0.1000	Do Not Reject H_0	0.0783	0.1000	Do Not Reject H_0

H_0 : Stationary; H_1 : Trend. The gray shading represents the supported laws/assets for this empirical study. RGM and NAD should be stationary to support Lehman's Laws of software evolution.

A.2 The KPSS Test and Hypotheses results (at FC)

Variable	Commonalities		Variabilities		Product-Specific	
	KPSS Test	p-value	Decision	KPSS Test	p-value	Decision
NA	0.0776	0.1000	Do Not Reject H_0	0.0916	0.1000	Do Not Reject H_0
LOC	1.0470	0.0100	Reject H_0	0.5130	0.0387	Reject H_0
NCLOC	0.2041	0.1000	Do Not Reject H_0	0.0854	0.1000	Do Not Reject H_0
NCM	0.1041	0.1000	Do Not Reject H_0	0.0851	0.1000	Do Not Reject H_0
RGM	-	-	-	-	-	-
NAD	0.0776	0.1000	Do Not Reject H_0	0.0916	0.1000	Do Not Reject H_0

H_0 : Stationary; H_1 : Trend. The gray shading represents the supported laws/assets for this empirical study. RGM and NAD should be stationary to support Lehman's Laws of software evolution.

B

SPL Evolution: A Systematic Mapping Study

This appendix presents some relevant data from the systematic mapping study for SPL evolution, earlier described in [Chapter 4](#).

B.1 Primary studies selected

- (S1) N. Abbas, J. Andersson, and W. Löwe. Autonomic software product lines (ASPL). In *European Conference on Software Architecture: Companion Volume (ECSA)*, 2010, pages 324–331.
- (S2) N. Abbas, J. Andersson, and D. Weyns. Knowledge evolution in autonomic software product lines. In *Software Product Line Conference (SPLC)*, 2011, pages 1–36.
- (S3) M. Acher, A. Cleve, P. Collet, P. Merle, L. Duchien, and P. Lahire. Reverse engineering architectural feature models. In *European Conference on Software Architecture (ECSA)*, pages 220–235, 2011.
- (S4) S. Ajila and A. Kaba. Using traceability mechanisms to support software product line evolution. In *International Conference on Information Reuse and Integration (IRI)*, 2004, pages 157 – 162.
- (S5) S. A. Ajila and A. B. Kaba. Evolution support mechanisms for software product line process. *Journal of Systems and Software (JSS)*, 2008 81(10):1784 – 1801.
- (S6) G. Alferez and V. Pelechano. Context-aware autonomous web services in software product lines. In *Software Product Line Conference (SPLC)*, 2011, pages 100 –109.
- (S7) V. Alves, R. Gheyi, T. Massoni, U. Kulesza, P. Borba, and C. J. P. de Lucena. Refactoring product lines. In *Generative Programming: Concepts and Experiences (GPCE)*, 2006, pages 201–210.
- (S8) V. Alves, P. M. Jr., L. Cole, P. Borba, and G. Ramalho. Extracting and evolving mobile games product lines. In *Software Product Lines Conference (SPLC)*, 2005, pages 70–81.
- (S9) M. Anastasopoulos. Increasing efficiency and effectiveness of software product line evolution: an infrastructure on top of configuration management. In *Joint international and annual ERCIM workshops on Principles of software evolution (IWPSE) and software evolution (Evol)*, 2009, pages 47–56.
- (S10) M. Anastasopoulos, T. H. B. de Oliveira, D. Muthig, E. S. de Almeida, and S. R. de Lemos Meira. Evolving a software product line reuse infrastructure: A configuration management solution. In *Variability Modelling of Software-intensive Systems (VaMoS)*, 2009, pages 19–28.

- (S11) J. Axelsson. Evolutionary architecting of embedded automotive product lines: An industrial case study. In Joint Working IEEE/IFIP Conference on Software Architecture, European Conference on Software Architecture (WICSA/ECSA), 2009, pages 101–110.
 - (S12) J. Axelsson. Improving the evolutionary architecting process for embedded system product lines. In Systems Conference (SysCon), 2011, pages 334–341.
 - (S13) J. Bayer, J.-F. Girard, M. Würthner, J.-M. DeBaud, and M. Apel. Transitioning legacy assets to a product line architecture. In European software engineering conference. International symposium on Foundations of software engineering (ESEC/FSE), 1999, pages 446–463.
 - (S14) N. Bencomo, G. S. Blair, C. A. Flores-Cortés, and P. Sawyer. Reflective component-based technologies to support dynamic variability. In Workshop on Variability Modelling of Software-intensive Systems (VaMoS), 2008, pages 141–150.
 - (S15) N. Bencomo, P. Sawyer, G. S. Blair, and P. Grace. Dynamically adaptive systems are product lines too: Using model-driven techniques to capture dynamic variability of adaptive systems. In Software Product Line Conference (SPLC), 2008, pages 23–32.
 - (S16) P. Borba, L. Teixeira, and R. Gheyi. A theory of software product line refinement. Theoretical Computer Science, International Colloquium on Theoretical Aspects of Computing. 2012, 455(0):2–30.
 - (S17) G. Botterweck, A. Pleuss, D. Dhungana, A. Polzer, and S. Kowalewski. Evofm: feature-driven planning of product-line evolutionary. In ICSE Workshop on Product Line Approaches in Software Engineering (PLEASE), 2010, pages 24–31.
 - (S18) G. Botterweck, A. Pleuss, A. Polzer, and S. Kowalewski. Towards feature-driven planning of product-line evolution. In International Workshop on Feature-Oriented Software Development (FOSD), 2009, pages 109–116.
 - (S19) H. Breivold, S. Larsson, and R. Land. Migrating industrial systems towards software product lines: Experiences and observations through case studies. In Euromicro Conference Software Engineering and Advanced Applications (SEAA), 2008, pages 232–239.
 - (S20) H. Brummermann, M. Keunecke, and K. Schmid. Formalizing distributed evolution of variability in information system ecosystems. In International Workshop on Variability Modeling of Software-Intensive Systems (VaMoS), 2012, pages 11–19.
-

-
- (S21) C. Cetina, P. Giner, J. Fons, and V. Pelechano. Designing and prototyping dynamic software product lines: Techniques and guidelines. In *Software Product Lines Conference (SPLC)*, 2010, pages 331–345.
- (S22) C. Cetina, V. Pelechano, P. Trinidad, and A. R. Cortés. An architectural discussion on dspl. In *Software Product Line Conference (SPLC)*, 2008, pages 59–68.
- (S23) Y. Chen, G. Gannod, J. Collofello, and H. Sarjoughian. Using simulation to facilitate the study of software product line evolution. In *International Workshop on Principles of Software Evolution*, 2004, pages 103–112.
- (S24) M. Cordy, A. Classen, P.-Y. Schobbens, P. Heymans, and A. Legay. Managing evolution in software product lines: a model-checking perspective. In *International Workshop on Variability Modeling of Software-Intensive Systems (VaMoS)*, 2012, pages 183–191.
- (S25) S. Creff, J. Champeau, J.-M. Jézéquel, and A. Monégier. Model-based product line evolution: an incremental growing by extension. In *Software Product Line Conference (SPLC)*, 2012, pages 107–114.
- (S26) F. Damiani, L. Padovani, and I. Schaefer. A formal foundation for dynamic delta-oriented software product lines. In *International Conference on Generative Programming and Component Engineering (GPCE)*, 2012, pages 1–10.
- (S27) F. Damiani and I. Schaefer. Dynamic delta-oriented programming. In *Software Product Line Conference (SPLC)*, 2011, pages 1–34.
- (S28) S. Deelstra, M. Sinnema, and J. Bosch. Variability assessment in software product families. *Information and Software Technology (IST)*, 2009, 51(1):195 – 218.
- (S29) D. Dhungana, P. Grünbacher, R. Rabiser, and T. Neumayer. Structuring the modeling space and supporting evolution in software product line engineering. *Journal of Systems and Software (JSS)*, 2010, 83(7):1108 – 1122.
- (S30) D. Dhungana, T. Neumayer, P. Grunbacher, and R. Rabiser. Supporting evolution in model-based product line engineering. In *Software Product Line Conference (SPLC)*, 2008, pages 319 –328.
- (S31) J. Díaz, J. Pérez, J. Garbajosa, and A. L. Wolf. Change impact analysis in product-line architectures. In *European Conference on Software Architecture (ECSA)*, 2011, pages 114–129.

- (S32) J. S. Dong, K. Lee, K. H. Kim, S. T. Kim, J. M. Cho, and T. H. Kim. Platform maintenance process for software quality assurance in product line. In *International Conference on Computer Science and Software Engineering*, 2008, pages 325–331.
 - (S33) C. Elsner, G. Botterweck, D. Lohmann, and W. Schröder-Preikschat. Variability in time - product line variability and evolution revisited. In *Variability Modelling of Software-intensive Systems (VaMoS)*, 2010, pages 131–137.
 - (S34) R. Froschauer, D. Dhungana, and P. Grunbacher. Managing the life-cycle of industrial automation systems with product line variability models. In *Euromicro Conference Software Engineering and Advanced Applications (SEAA)*, 2008, pages 35–42.
 - (S35) N. Gámez and L. Fuentes. Software product line evolution with cardinality-based feature models. In *International Conference on Software Reuse (ICSR)*, 2011, pages 102–118.
 - (S36) N. Gamez and L. Fuentes. Architectural evolution of famiware using cardinality-based feature models. *Information and Software Technology (IST)*, 2012, 55(3):563–580.
 - (S37) A. Garg, M. Critchlow, P. Chen, C. Van der Westhuizen, and A. van der Hoek. An environment for managing evolving product line architectures. In *International Conference on Software Maintenance (ICSM)*, 2003, pages 358–367.
 - (S38) H. Gomaa and K. Hashimoto. Dynamic software adaptation for service-oriented product lines. In *Software Product Line Conference (SPLC)*, 2011, pages 1–8.
 - (S39) H. Gomaa and M. Hussein. Software reconfiguration patterns for dynamic evolution of software architectures. In *Working IEEE/IFIP Conference on Software Architecture (WICSA)*, 2004, pages 79–88.
 - (S40) H. Gomaa and M. Hussein. Model-based software design and adaptation. In *International Workshop on Software Engineering for Adaptive and Self-Managing Systems (ICSE-SEAMS)*, 2007, pages 1–7.
 - (S41) W. Heider, R. Rabiser, P. Grunbacher. Facilitating the evolution of products in product line engineering by capturing and replaying configuration decisions. *International Journal on Software Tools for Technology Transfer*, 2012, 14:613–630.
 - (S42) S. Günther and S. Sunkle. Dynamically adaptable software product lines using ruby metaprogramming. In *International Workshop on Feature-Oriented Software Development (FOSD)*, 2010, pages 80–87.
-

-
- (S43) J. Guo, Y. Wang, P. Trinidad, and D. Benavides. Consistency maintenance for evolving feature models. *Expert Systems with Applications*, 2012, 39(5):4987 – 4998.
- (S44) S. O. Hallsteinsen, E. Stav, A. Solberg, and J. Floch. Using product line techniques to build adaptive systems. In *Software Product Line Conference (SPLC)*, 2006, pages 141–150.
- (S45) G. Hanssen, A. Yamashita, R. Conradi, and L. Moonen. Software entropy in agile product evolution. In *Hawaii International Conference on System Sciences (HICSS)*, 2010, pages 1–10.
- (S46) W. Heider, R. Froschauer, P. Grünbacher, R. Rabiser, and D. Dhungana. Simulating evolution in model-based product line engineering. *Information and Software Technology (IST)*, 2010, 52(7):758 – 769.
- (S47) W. Heider, R. Rabiser, P. Grünbacher, and D. Lettner. Using regression testing to analyze the impact of changes to variability models on products. In *Software Product Line Conference (SPLC)*, 2012, pages 196–205.
- (S48) M. Helvensteijn. Dynamic delta modeling. In *Software Product Line Conference (SPLC)*, 2012, pages 127–134.
- (S49) M. Inoki and Y. Fukazawa. Software product line evolution method based on kaizen approach. In *Proceedings of the ACM symposium on Applied computing (SAC)*, 2007, pages 1207–1214.
- (S50) M. Jahn, R. Rabiser, P. Grunbacher, M. Loberbauer, R. Wolfinger, and H. Mossenbock. Supporting model maintenance in component-based product lines. In *Working IEEE/IFIP Conference on Software Architecture (WICSA) and European Conference on Software Architecture (ECSA)*, 2012, pages 21 –30.
- (S51) K. C. Kang, M. Kim, J. Lee, and B. Kim. Feature-oriented re-engineering of legacy systems into product line assets - a case study. In *Software Product Line Conference (SPLC)*, 2005, pages 45–56.
- (S52) C. Kästner, S. Apel, and M. Kuhlemann. A model of refactoring physically and virtually separated features. In *Generative Programming: Concepts and Experiences (GPCE)*, 2009, pages 157–166.

- (S53) K. Kim, H. Kim, and W. Kim. Building software product line from the legacy systems "experience in the digital audio and video domain". In Software Product Line Conference (SPLC), 2007, pages 171–180.
- (S54) M. Kim, J.-H. Kim, and S. Park. Tool support for quality evaluation and feature selection to achieve dynamic quality requirements change in product lines. In Software Product Line Conference (SPLC), 2008, pages 69–78.
- (S55) M. Kim, S. Park, and J. Lee. An approach to dynamically achieving quality requirements change in product line engineering. In Software Product Line Conference (SPLC), 2007, pages 41–50.
- (S56) J. Knodel, I. John, D. Ganesan, M. Pinzger, F. Usero, J. L. Arciniegas, and C. Riva. Asset recovery and their incorporation into product lines. In Working Conference on Reverse Engineering (WCRE), 2005, pages 120–129.
- (S57) S. Krishnan, R. R. Lutz, and K. Goševa-Popstojanova. Empirical evaluation of reliability improvement in an evolving software product line. In Working Conference on Mining Software Repositories (MSR), 2011, pages 103–112.
- (S58) S. Krishnan, C. Strasburg, R. R. Lutz, and K. Goševa-Popstojanova. Are change metrics good predictors for an evolving software product line? In Conference on Predictive Models in Software Engineering (Promise), 2011, pages 1–7.
- (S59) M. Kuhlemann, D. S. Batory, and S. Apel. Refactoring feature modules. In International Conference on Software Reuse (ICSR), 2009, pages 106–115.
- (S60) J. Lee and D. Muthig. Feature-oriented analysis and specification of dynamic product reconfiguration. In International Conference Software Reuse (ICSR), 2008, pages 154–165.
- (S61) J. Liu, J. Dehlinger, H. Sun, and R. Lutz. State-based modeling to support the evolution and maintenance of safety-critical software product lines. In International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS), 2007, pages 596–608.
- (S62) F. Loesch and E. Ploedereder. Restructuring variability in software product lines using concept analysis of product configurations. In European Conference on Software Maintenance and Reengineering (CSMR), 2007, pages 159–170.
-

-
- (S63) N. López, R. Casallas, and A. van der Hoek. Issues in mapping change-based product line architectures to configuration management systems. In *Software Product Line Conference (SPLC)*, 2009, pages 21–30.
- (S64) R. Lopez-Herrejon, L. Montalvillo-Mendizabal, and A. Egyed. From requirements to features: An exploratory study of feature-oriented refactoring. In *Software Product Line Conference (SPLC)*, 2011, pages 181–190.
- (S65) R. Lotufo, S. She, T. Berger, K. Czarnecki, and A. Wasowski. Evolution of the linux kernel variability model. In *Software Product Line Conference (SPLC)*, 2010, pages 136–150.
- (S66) F. G. Marinho, R. M. Andrade, C. Werner, W. Viana, M. E. Maia, L. S. Rocha, E. Teixeira, J. B. F. Filho, V. L. Dantas, F. Lima, and S. Aguiar. Mobiline: A nested software product line for the domain of mobile and context-aware applications. *Science of Computer Programming*, 2012, 78(12):2381–2398.
- (S67) T. Mende, F. Beckwermert, R. Koschke, and G. Meier. Supporting the grow-and-prune model in software product lines evolution using clone detection. In *European Conference on Software Maintenance and Reengineering (CSMR)*, 2008, pages 163–172.
- (S68) B. Michalik, D. Weyns, N. Boucke, and A. Helleboogh. Supporting online updates of software product lines: A controlled experimental. In *Empirical Software Engineering and Measurement (ESEM)*, 2011, pages 187–196.
- (S69) I. Montero, J. Peña, and A. R. Cortés. Business family engineering - managing the evolution of business driven systems. In *Software Product Line Conference (SPLC)*, 2007, pages 33–40.
- (S70) L. Neves, L. Teixeira, D. Sena, V. Alves, U. Kulezsa, and P. Borba. Investigating the safe evolution of software product lines. In *Proceedings of the ACM international conference on Generative programming and component engineering (GPCE)*, 2011, pages 33–42.
- (S71) N. Niu, J. Savolainen, and Y. Yu. Variability modeling for product line viewpoints integration. In *Computer Software and Applications Conference (COMPSAC)*, 2010, pages 337–346.
- (S72) C. Nunes, A. Garcia, C. J. P. de Lucena, and J. Lee. History-sensitive heuristics for recovery of features in code of evolving program families. In *Software Product Lines Conference (SPLC)*, 2012, pages 136–145.

- (S73) O. Ortiz, A. B. García, R. Capilla, J. Bosch, and M. Hinchey. Runtime variability for dynamic reconfiguration in wireless sensor network product lines. In *Software Product Line Conference (SPLC)*, 2012, pages 143–150.
- (S74) C. Parra, X. Blanc, A. Cleve, and L. Duchien. Unifying design and runtime software adaptation using aspect models. *Science of Computer Programming*, 2011, 76(12):1247–1260.
- (S75) C. Parra, X. Blanc, and L. Duchien. Context awareness for dynamic service-oriented product lines. In *Software Product Line Conference (SPLC)*, 2009, pages 131–140.
- (S76) T. Patzke, M. Becker, M. Steffens, K. Sierszecki, J. E. Savolainen, and T. Fogdal. Identifying improvement potential in evolving product line infrastructures: 3 case studies. In *Software Product Line Conference (SPLC)*, 2012, pages 239–248.
- (S77) X. Peng, Y. Yu, and W. Zhao. Analyzing evolution of variability in a software product line: From contexts and requirements to features. *Information and Software Technology*, 2011, 53(7):707–721.
- (S78) G. Perrouin, F. Chauvel, J. DeAntoni, and J.-M. Jézéquel. Modeling the variability space of self-adaptive applications. In *Software Product Line Conference (SPLC)*, 2008, pages 15–22.
- (S79) J. Peña, M. G. Hinchey, M. Resinas, R. Sterritt, and J. L. Rash. Designing and managing evolving systems using a mas product line approach. *Science of Computer Programming*, 2007, 66(1):71–86.
- (S80) A. Pleuss, G. Botterweck, D. Dhungana, A. Polzer, and S. Kowalewski. Model-driven support for product line evolution on feature level. *Journal of Systems and Software (JSS)*, 2012, 85(10):2261–2274.
- (S81) M. Ribeiro and P. Borba. Improving guidance when restructuring variabilities in software product lines. In *European Conference on Software Maintenance and Reengineering (CSMR)*, 2009, pages 79–88.
- (S82) M. Ribeiro, F. Queiroz, P. Borba, T. Tolêdo, C. Brabrand, and S. Soares. On the impact of feature dependencies when maintaining preprocessor-based software product lines. In *Generative programming and component engineering (GPCE)*, 2011, pages 23–32.
-

-
- (S83) C. Riva and C. Del Rosso. Experiences with software product family evolution. In Workshop on Principles of Software Evolution, 2003, pages 161–169.
- (S84) K. Romanovsky, D. Koznov, and L. Minchin. Refactoring the documentation of software product lines. In Central and East European conference on Software engineering techniques (CEE-SET), 2011, pages 158–170.
- (S85) M. Rosenmüller, N. Siegmund, S. Apel, and G. Saake. Flexible feature binding in software product lines. *Automated Software Engineering*, 2011, 18(2):163–197.
- (S86) M. Rosenmüller, N. Siegmund, M. Pukall, and S. Apel. Tailoring dynamic software product lines. In Generative Programming and Component Engineering (GPCE), 2011, pages 3–12.
- (S87) E. Roubtsova and S. Roubtsov. Behavioural inheritance in the uml to model software product lines. *Science of Computer Programming*, 2004, 53(3):409–434.
- (S88) K. Saller, S. Oster, A. Schurr, J. Schroeter, and M. Lochau. Reducing feature models to improve runtime adaptivity on resource limited devices. In Software Product Line Conference (SPLC), 2012, pages 135–142.
- (S89) D. Saraiva, L. Pereira, T. V. Batista, F. C. Delicato, P. F. Pires, U. Kulesza, R. Araújo, T. Freitas, S. M. Filho, and A. L. S. Souto. Architecting a model-driven aspect-oriented product line for a digital tv middleware: A refactoring experiences. In European Conference on Software Architecture (ECSA), 2010, pages 166–181.
- (S90) J. Savolainen and J. Kuusela. Volatility analysis framework for product lines. In Proceedings of the symposium on Software reusability: putting software reuse in context (SSR), 2001, pages 133–141.
- (S91) S. R. Schach and A. Tomer. Development/maintenance/reuse: software evolution in product lines. In Software Product Line Conference (SPLC), 2000, pages 437–450.
- (S92) K. Schmid and H. Eichelberger. From static to dynamic software product lines. In Software Product Line Conference (SPLC), 2008, pages 33–38.
- (S93) J. Schroeter, P. Mucha, M. Muth, K. Jugel, and M. Lochau. Dynamic configuration management of cloud-based applications. In Software Product Line Conference (SPLC), 2012, pages 171–178.

- (S94) M. Schubanz, A. Pleuss, G. Botterweck, and C. Lewerentz. Modeling rationale over time to support product line evolution planning. In *Workshop on Variability Modeling of Software-Intensive Systems (VaMoS)*, 2012, pages 193–199.
- (S95) S. Schulze, T. Thüm, M. Kuhlemann, and G. Saake. Variant-preserving refactoring in feature-oriented software product lines. In *Workshop on Variability Modeling of Software-Intensive Systems (VaMoS)*, 2012, pages 73–81.
- (S96) C. Seidl, F. Heidenreich, and U. Assmann. Co-evolution of models and feature mapping in software product lines. In *Software Product Line Conference (SPLC)*, 2012, pages 76–85.
- (S97) L. Shen, X. Peng, J. Liu, and W. Zhao. Towards feature-oriented variability reconfiguration in dynamic software product lines. In *International Conference on Software Reuse (ICSR)*, 2011, pages 52–68.
- (S98) L. Shen, X. Peng, J. Zhu, and W. Zhao. Synchronized architecture evolution in software product line using bidirectional transformation. In *Computer Software and Applications Conference (COMPSAC)*, 2010, pages 389–394.
- (S99) H. Shokry and M. A. Babar. Dynamic software product line architectures using service-based computing for automotive systems. In *Software Product Line Conference (SPLC)*, 2008, pages 53–58.
- (S100) D. Simon and T. Eisenbarth. Evolutionary introduction of software product lines. In *Software Product Line Conference (SPLC)*, 2002, pages 272–282.
- (S101) D. B. Smith, L. O’Brien, and J. Bergey. Using the options analysis for reengineering (oar) method for mining components for a product line. In *Software Product Line Conference (SPLC)*, 2002, pages 316–327.
- (S102) N. Anquetil, U. Kulesza, R. Mitschke, A. Moreira, J. Royer, A. Rummler, and A. Sousa. A model-driven traceability framework for software product lines. *Software & Systems Modeling*, 2010, 9(4)427–451.
- (S103) C. Thao, E. Munson, and T. Nguyen. Software configuration management for product derivation in software product families. In *International Conference and Workshop on the Engineering of Computer Based Systems (ECBS)*, 2008, pages 265–274.
- (S104) T. Thum, D. Batory, and C. Kastner. Reasoning about edits to feature models. In *International Conference on Software Engineering (ICSE)*, 2009, pages 254–264.
-

-
- (S105) L. P. Tizzei, M. Dias, C. M. Rubira, A. Garcia, and J. Lee. Components meet aspects: Assessing design stability of a software product line. *Information and Software Technology (IST)*, 2011, 53(2):121–136.
- (S106) P. Trinidad, A. R. Cortés, J. Peña, and D. Benavides. Mapping feature models onto component models to build dynamic software product lines. In *Software Product Line Conference (SPLC)*, 2007, pages 51–56.
- (S107) M. I. Ullah, G. Ruhe, and V. Garousi. Decision support for moving from a single product to a product portfolio in evolving software systems. *Journal of Systems and Software (JSS)*, 2010, 83(12):2496–2512.
- (S108) M. T. Valente, V. Borges, and L. T. Passos. A semi-automatic approach for extracting software product lines. *IEEE Transactions on Software Engineering (TSE)*, 2012, 38(4):737–754.
- (S109) D. Weyns and B. Michalik. Codifying architecture knowledge to support online evolution of software product lines. In *Workshop on SHaring and Reusing Architectural Knowledge (SHARK)*, 2011, pages 37–44.
- (S110) D. Weyns, B. Michalik, A. Helleboogh, and N. Boucke. An architectural approach to support online updates of software product lines. In *IEEE/IFIP Conference on Software Architecture (WICSA)*, 2011, pages 204–213.
- (S111) R. Wolfinger, S. Reiter, D. Dhungana, P. Grunbacher, and H. Prafhofer. Supporting runtime system adaptation through product line engineering and plug-in techniques. In *International Conference on Composition-Based Software Systems (ICCBSS)*, 2008, pages 21–30.
- (S112) Y. Wu, Y. Yang, X. Peng, C. Qiu, and W. Zhao. Recovering object-oriented framework for software product line reengineering. In *International Conference on Software Reuse (ICSR)*, 2011, pages 119–134.
- (S113) Y. Wu, D. Zowghi, X. Peng, and W. Zhao. Towards understanding requirement evolution in a software product line an industrial case study. In *International Workshop on the Twin Peaks of Requirements and Architecture (Twin Peaks)*, 2012, pages 7–14.
- (S114) Y. Xue, Z. Xing, and S. Jarzabek. Understanding feature evolution in a family of product variants. In *Working Conference on Reverse Engineering (WCRE)*, 2010, pages 109–118.

- (S115) Y. Xue, Z. Xing, and S. Jarzabek. Feature location in a collection of product variants. In Working Conference on Reverse Engineering (WCRE), 2012, pages 145–154.
 - (S116) K. Yoshimura, F. Narisawa, K. Hashimoto, and T. Kikuno. Fave: factor analysis based approach for detecting product line variability from change history. In International working conference on Mining software repositories (MRS), 2008, pages 11–18.
 - (S117) G. Zhang, L. Shen, X. Peng, Z. Xing, and W. Zhao. Incremental and iterative reengineering towards software product line: An industrial case study. In International Conference on Software Maintenance (ICSM), 2011, pages 418–427.
 - (S118) W. Zhang, S. Jarzabek, N. Loughran, and A. Rashid. Reengineering a pc-based system into the mobile device product line. In International Workshop on Principles of Software Evolution (IWPSE), 2003, pages 149–160.
 - (S119) M. Acher, B. Baudry, P. Heymans, A. Cleve, J.-L. Hainaut. Support for reverse engineering and maintaining feature models. In International Workshop on Variability Modelling of Software-intensive Systems (VaMoS), 2013, pages 1–8.
 - (S120) M. Acher, A. Cleve, P. Collet, P. Merle, L. Duchien, P. Lahire. Extraction and evolution of architectural variability models in plugin-based systems. *Software And Systems Modeling*, 2014, 13(4):1367–1394.
 - (S121) S. Adelsberger, S. Sobernig, G. Neumann. Towards assessing the complexity of object migration in dynamic, feature-oriented software product lines. In International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS), 2013, pages 1–8.
 - (S122) A. Benlarabi. Towards a co-evolution model for software product lines based on cladistics. In International Conference on Research Challenges in Information Science (RCIS), 2014, pages 1–6.
 - (S123) C. Cetina, P. Giner, J. Fons, V. Pelechano, Prototyping dynamic software product lines to evaluate run-time reconfigurations, *Science of Computer Programming*, 2013, 78(12):2399–2413.
 - (S124) N. Dintzner, A. Van Deursen, M. Pinzger. Extracting feature model changes from the linux kernel using fmdiff. In International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS), 2013, pages 1–8, 2013.
-

-
- (S125) N. Gámez, L. Fuentes. Architectural evolution of famiware using cardinality-based feature models. *Information And Software Technology (IST)*, 2013, 55(3):563–580.
- (S126) F. N. Gaia, G. C. S. Ferreira, E. Figueiredo, M. de Almeida Maia. A quantitative and qualitative assessment of aspectual feature modules for evolving software product lines. *Science of Computer Programming*, 2014, 96(2),230–253.
- (S127) R. Hellebrand, A. Silva, M. Becker, B. Zhang, K. Sierszecki, J. Savolainen. Coevolution of variability models and code: An industrial case study. In *Software Product Line Conference (SPLC)*, 2014, pages 274–283.
- (S128) T. Kanda, T. Ishio, K. Inoue, Extraction of product evolution tree from source code of product variants. In *Software Product Line Conference (SPLC)*, 2013, pages 141–150.
- (S129) J. Koscielny, S. Holthusen, I. Schaefer, S. Schulze, L. Bettini, F. Damiani. Deltaj 1.5: Delta-oriented programming for java 1.5. In *International Conference on Principles and Practices of Programming on the Java Platform: Virtual Machines, Languages, and Tools (PPPJ)*, 2014, pages 63–74.
- (S130) S. Krishnan, C. Strasburg, R. R. Lutz, K. Goseva-Popstojanova, K. S. Dorman. Predicting failure-proneness in an evolving software product line. *Information and Software Technology (IST)*, 2013, 55(8):1479–1495.
- (S131) R. E. L. Herrejon, L. Linsbauer, J. A. Galindo, J. A. Parejo, D. Benavides, S. Segura, A. Egyed. An assessment of search-based techniques for reverse engineering feature models, *Journal of Systems and Software (JSS)*, 2014, 103:353–369.
- (S132) A. Murguzur, R. Capilla, S. Trujillo, O. Ortiz, R. E. Lopez-Herrejon. Context variability modeling for runtime configuration of service-based dynamic software product lines. In *International Software Product Line Conference (SPLC)*, 2014, pages 2–9.
- (S133) R. Muschevici, D. Clarke, J. Proenca. Executable modelling of dynamic software product lines in the abs language. In *International Workshop on Feature-Oriented Software Development (FOSD)*, 2013, pages 17–24.
- (S134) L. T. Passos, J. Guo, L. Teixeira, K. Czarnecki, A. Wasowski, P. Borba. Coevolution of variability models and related artifacts: a case study from the linux kernel. In *International Software Product Line Conference (SPLC)*, 2013, pages 91–100.

- (S135) C. Quinton, A. Pleuss, D. L. Berre, L. Duchien, G. Botterweck. Consistency checking for the evolution of cardinality-based feature models. In *International Software Product Line Conference (SPLC)*, 2014, pages 122–131.
- (S136) D. Romero, S. Urli, C. Quinton, M. Blay-Fornarino, P. Collet, L. Duchien, S. Mosser. Splemma: A generic framework for controlled-evolution of software product lines. In *International Software Product Line Conference Co-located Workshops (SPLC)*, 2013, pages 59–66.
- (S137) S. Schulze, M. Lochau, S. Brunswig. Implementing refactorings for fop: Lessons learned and challenges ahead. In *International Workshop on Feature-Oriented Software Development (FOSD)*, 2013, pages 33–40.
- (S138) S. Schulze, O. Richers, I. Schaefer. Refactoring delta-oriented software product lines. In *International Conference on Aspect-oriented Software Development (AOSD)*, 2013, pages 73–84.
- (S139) C. Seidl, U. Assmann. Towards modeling and analyzing variability in evolving software ecosystems. In *International Workshop on Variability Modelling of Software-intensive Systems (VaMoS)*, 2013, pages 3:1–3:8.
- (S140) C. Seidl, I. Schaefer, U. Assmann. Integrated management of variability in space and time in software families. In *International Software Product Line Conference (SPLC)*, 2014, pages 22–31.
- (S141) J. White, J. A. Galindo, T. Saxena, B. Dougherty, D. Benavides, D. C. Schmidt. Evolving feature model configurations in software product lines. *Journal of Systems and Software (JSS)*, 2014, 87(0):119–136.
- (S142) B. Zhang, M. Becker, T. Patzke, K. Sierszecki, J. E. Savolainen. Variability evolution and erosion in industrial product lines: A case study. In *International Software Product Line Conference (SPLC)*, 2013, pages 168–177.

B.2 Data Extraction Form

1.Categories of SPL Requirement Evolution

- o New product family
- o Introduction of New Product
- o Adding New Features
- o Extend Standards Support
- o New Version of Infrastructure
- o Improvement of Quality Attribute

2.Temporal Properties Evaluated (When)

- o Time of change
 - [Static][Load Time][Dynamic/Run-Time]
- o Change history
 - [Sequential][Parallel]
- o Change frequency
 - [Continuous][Periodically][Arbitrary]
- o Anticipation
 - [Yes-Which?][No]

continued on next page

<i>continued from previous page</i>
3.Object of Change Evaluated (Where)
<ul style="list-style-type: none"> o Artifact <ul style="list-style-type: none"> [Core Asset Base][Core Asset][SPL Architecture][Product Architecture][Product][Other] o Granularity <ul style="list-style-type: none"> [Coarse Grained][Fine Grained] o Impact <ul style="list-style-type: none"> [Local][Global] o Change propagation <ul style="list-style-type: none"> [Change Impact Analysis][Traceability Analysis][Effort Estimation]
4.System Properties (What)
<ul style="list-style-type: none"> o Availability <ul style="list-style-type: none"> [System Always Available][System not Always Available] o Activeness <ul style="list-style-type: none"> [Reactive][Proactive] o Openness <ul style="list-style-type: none"> [Open][Closed] o Safety <ul style="list-style-type: none"> [Static][Dynamic] o Other
<i>continued on next page</i>

continued from previous page

5.Change Support (How)

- o Degree of automation
[Automated][Partially][Manual]
- o Degree of formality
[Ad hoc][Mathematical Formalism]
- o Change type
[Structural (Re-engineering)][Semantics (Refactoring)]

6.Phase of the SPL life cycle in which the evolution is applied

- o Domain Engineering
[Scoping][Requirement][Architecture][Realization][Test]
- o Application Engineering
[Requirement][Architecture][Realization][Test]

7.Evaluation Procedure

- o Type of Evaluation
[Case Studies][Surveys][Controlled Experiments][Feasibility Study][Not Evaluated]

8.Tool Support

- o Automatic
- o Manual

continued on next page

<i>continued from previous page</i>
9.Current Usage
<ul style="list-style-type: none">o Academiao Industry

B.3 Search String for each Electronic Database

IEEE Xplore	
Restricted to: (Content Type) Conference Publications, Journals and Magazines	
Title	Search String: (((("Document Title":evol* OR "Document Title":maint* OR "Document Title":chang* OR "Document Title":modif*) AND (p_Title:"product line" OR "Document Title": "product-line" OR "Document Title": "product family" OR "Document Title": "product-family" OR "Document Title": "product families" OR "Document Title": "product-families" OR "Document Title":SPL OR "Document Title": "family of product" OR "Document Title": "families of product"))))
Abstract	Search String: (("Abstract":evol* OR "Abstract":maint* OR "Abstract":chang* OR "Abstract":modif*) AND (p_Abstract:"product line" OR "Abstract": "product-line" OR "Abstract": "product family" OR "Abstract": "product-family" OR "Abstract": "product families" OR "Abstract": "product-families" OR "Abstract":SPL OR "Abstract": "family of product" OR "Abstract": "families of product"))
Keywords	Search String: (((("Author Keywords":evol* OR "Author Keywords":maint* OR "Author Keywords":chang* OR "Author Keywords":modif*) AND (p_Author_Terms:"product line" OR "Author Keywords": "product-line" OR "Author Keywords": "product family" OR "Author Keywords": "product-family" OR "Author Keywords": "product families" OR "Author Keywords": "product-families" OR "Author Keywords":SPL OR "Author Keywords": "family of product" OR "Author Keywords": "families of product"))))

B.3. SEARCH STRING FOR EACH ELECTRONIC DATABASE

ACM Digital Library	
Restricted to: Journal Proceeding Transaction Magazine	
Title	Search String: (Title:evol* or Title:maint* or Title:chang* or Title:modif*) and (Title:"product line" or Title:"product-line" or Title:"product family" or Title:"product-family" or Title:"product families" or Title:"product-families" or Title:"spl" or Title:"family of product" or Title:"families of product")
Abstract	Search String: (Abstract:evol* or Abstract:maint* or Abstract:chang* or Abstract:modif*) and (Abstract:"product line" or Abstract:"product-line" or Abstract:"product family" or Abstract:"product-family" or Abstract:"product families" or Abstract:"product-families" or Abstract:"spl" or Abstract:"family of product" or Abstract:"families of product")
Keywords	Search String: (Keywords:evol* or Keywords:maint* or Keywords:chang* or Keywords:modif*) and (Keywords:"product line" or Keywords:"product-line" or Keywords:"product family" or Keywords:"product-family" or Keywords:"product families" or Keywords:"product-families" or Keywords:"spl" or Keywords:"family of product" or Keywords:"families of product")

SpringerLink	
Restricted to: Conferences and journals, Papers in English, Computer Science, Software Engineering	
General	Search String: (evol* OR maint* OR chang* OR modif*) AND ("product line" OR "product-line" OR "product family" OR "product-family" OR "product families" OR "product-families" OR SPL or "family of product" or "families of product")

Science Direct	
Restricted to: Computer Science	
Title	Search String: TITLE((evol* OR maint* OR chang* OR modif*) AND (product AND line OR product-line OR product AND family OR product-family OR product AND families OR product-families OR SPL OR family AND of AND product OR families AND of AND product))
Abstract	Search String: ABSTRACT((evol* OR maint* OR chang* OR modif*) AND (product AND line OR product-line OR product AND family OR product-family OR product AND families OR product-families OR SPL OR family AND of AND product OR families AND of AND product))
Keywords	Search String: KEYWORDS((evol* OR maint* OR chang* OR modif*) AND (product AND line OR product-line OR product AND family OR product-family OR product AND families OR product-families OR SPL OR family AND of AND product OR families AND of AND product))

B.4 Mapping of the primary studies

ID	RQ1.1						RQ1.2						RQ1.3						RQ1.4																	
	a	b	c	d	e	f	a	b	c	d	e	f	g	h	i	a	b	c	d	e	f	g	h	i	j	k	l	m	a	b	c	d	e	f	g	h
(S1)	X						X									X												X								
(S2)	X						X									X													X							
(S3)		X						X								X													X							
(S4)		X					X									X													X							
(S5)		X						X								X													X							
(S6)		X						X								X													X							
(S7)		X						X								X													X							
(S8)	X						X									X												X								
(S9)		X					X									X												X								
(S10)		X					X									X											X									
(S11)						X										X											X									
(S12)						X										X											X									
(S13)	X						X									X											X									
(S14)		X					X									X											X									
(S15)		X					X									X											X									
(S16)		X					X									X											X									
(S17)							X									X											X									
(S18)							X									X											X									
(S19)	X						X									X											X									
(S20)		X					X									X											X									
(S21)		X					X									X											X									
(S22)		X					X									X											X									
(S23)		X					X									X											X									
(S24)		X					X									X											X									
(S25)		X					X									X											X									
(S26)		X					X									X											X									
(S27)		X					X									X											X									
(S28)		X					X									X											X									
(S29)		X					X									X											X									
(S30)		X					X									X											X									
(S31)		X					X									X											X									
(S32)		X					X									X											X									
(S33)		X					X									X											X									
(S34)		X					X									X											X									

continued on next page

RQ1.1: (a) New Product Family; (b) Introduction of New Product; (c) Adding New Features; (d) Extend Standards Support; (e) New Version of Infrastructure; (f) Improvement of Quality Attribute.

RQ1.2: (a) Static; (b) Load Time; (c) Dynamic Run-Time; (d) Parallel; (e) Sequential; (f) Arbitrary; (g) Continuous; (h) Periodically; (i) Anticipation.

RQ1.3: (a) Core Asset Base; (b) Core Asset; (c) SPL Architecture; (d) Product Architecture; (e) Product; (f) Other; (g) Fine Grained; (h) Coarse Grained; (i) Local; (j) Global; (k) Change Impact Analysis; (l) Effort Estimation; (m) Traceability Analysis.

RQ1.4: (a) System nor Always Available; (b) System Always Available; (c) Reactive; (d) Proactive; (e) Closed; (f) Open; (g) Static; (h) Dynamic.

B.4. MAPPING OF THE PRIMARY STUDIES

ID	continued from previous page												continued on next page											
	RQ1.1				RQ1.2				RQ1.3				RQ1.4											
	a	b	c	d	a	b	c	d	a	b	c	d	a	b	c	d	e	f	g	h				
(S35)	X				X				X				X											
(S36)	X				X				X				X											
(S37)	X				X				X				X											
(S38)	X				X				X				X											
(S39)	X				X				X				X											
(S40)	X				X				X				X											
(S41)	X				X				X				X											
(S42)	X				X				X				X											
(S43)	X				X				X				X											
(S44)	X				X				X				X											
(S45)	X				X				X				X											
(S46)	X				X				X				X											
(S47)	X				X				X				X											
(S48)	X				X				X				X											
(S49)	X				X				X				X											
(S50)	X				X				X				X											
(S51)	X				X				X				X											
(S52)	X				X				X				X											
(S53)	X				X				X				X											
(S54)	X				X				X				X											
(S55)	X				X				X				X											
(S56)	X				X				X				X											
(S57)	X				X				X				X											
(S58)	X				X				X				X											
(S59)	X				X				X				X											
(S60)	X				X				X				X											
(S61)	X				X				X				X											
(S62)	X				X				X				X											
(S63)	X				X				X				X											
(S64)	X				X				X				X											
(S65)	X				X				X				X											
(S66)	X				X				X				X											
(S67)	X				X				X				X											
(S68)	X				X				X				X											
(S69)	X				X				X				X											
(S70)	X				X				X				X											
(S71)	X				X				X				X											
(S72)	X				X				X				X											
(S73)	X				X				X				X											
(S74)	X				X				X				X											
(S75)	X				X				X				X											

RQ1.1: (a) New Product Family; (b) Introduction of New Product; (c) Adding New Features; (d) Extend Standards Support; (e) New Version of Infrastructure; (f) Improvement of Quality Attribute.
RQ1.2: (a) Static; (b) Load Time; (c) Dynamic Run-Time; (d) Parallel; (e) Sequential; (f) Arbitrary; (g) Continuous; (h) Periodically; (i) Anticipation.
RQ1.3: (a) Core Asset Base; (b) Core Asset; (c) SPL Architecture; (d) Product Architecture; (e) Product; (f) Other; (g) Fine Grained; (h) Course Grained; (i) Local; (j) Global; (k) Change Impact Analysis; (l) Effort Estimation; (m) Traceability Analysis.
RQ1.4: (a) System not Always Available; (b) System Always Available; (c) Reactive; (d) Proactive; (e) Closed; (f) Open; (g) Static; (h) Dynamic.

ID	RQ1.5							RQ1.6							RQ1.7					RQ1.8		RQ1.9		Q.A.					
	a	b	c	d	e	f	g	a	b	c	d	e	f	g	h	i	a	b	c	d	e	f	a		b	a	b		
(S9)					X			X								X							X		X			3	
(S10)			X	X	X			X							X									X		X			0
(S11)	X		X	X	X			X							X									X		X			-1
(S12)	X		X	X	X			X							X									X		X			-1
(S13)	X		X	X	X			X							X									X		X			0
(S14)	X		X	X	X			X							X									X		X			0
(S15)			X	X	X			X							X									X		X			1
(S16)	X		X	X	X			X							X									X		X			4
(S17)	X		X	X	X			X							X									X		X			4
(S18)	X		X	X	X			X							X									X		X			3
(S19)	X		X	X	X			X							X									X		X			X
(S20)	X		X	X	X			X							X									X		X			-1
(S21)	X		X	X	X			X							X									X		X			2
(S22)	X		X	X	X			X							X									X		X			2
(S23)	X		X	X	X			X							X									X		X			2
(S24)	X		X	X	X			X							X									X		X			1
(S25)			X	X	X			X							X									X		X			-2
(S26)	X		X	X	X			X							X									X		X			1
(S27)	X		X	X	X			X							X									X		X			1
(S28)	X		X	X	X			X							X									X		X			-1
(S29)	X		X	X	X			X							X									X		X			0
(S30)			X	X	X			X							X									X		X			3
(S31)			X	X	X			X							X									X		X			3
(S32)			X	X	X			X							X									X		X			3
(S33)	X		X	X	X			X							X									X		X			-1
(S34)			X	X	X			X							X									X		X			0
(S35)			X	X	X			X							X									X		X			-2
(S36)			X	X	X			X							X									X		X			-1
(S37)			X	X	X			X							X									X		X			-2
(S38)			X	X	X			X							X									X		X			-1
(S39)			X	X	X			X							X									X		X			-2
(S40)			X	X	X			X							X									X		X			2
(S41)			X	X	X			X							X									X		X			1
(S42)			X	X	X			X							X									X		X			0
(S43)			X	X	X			X							X									X		X			3
(S44)	X		X	X	X			X							X									X		X			0
(S45)			X	X	X			X							X									X		X			0
(S46)			X	X	X			X							X									X		X			0
(S47)			X	X	X			X							X									X		X			0
(S48)	X		X	X	X			X							X									X		X			-3

RQ1.5:(a) Manual; (b) Partially; (c) Automated; (d) Ad hoc; (e) Mathematical Formalism; (f) Structural (Re-engineering); (g) Semantics (Refactoring).

RQ1.6:(a) Scoping; (b) Requirement; (c) Architecture; (d) Realization; (e) Test; (f) Requirement; (g) Architecture; (h) Realization; (i) Test.

RQ1.7:(a) Not Evaluated; (b) Feasibility Study; (c) Controlled Experiments; (d) Case Studies; (e) Surveys; (f) Other.

RQ1.8:(a) Automatic; (b) Manual.

RQ1.9:(a) Academia; (b) Industry.

Q.A.: Sum of the Values from the Quality Assessment Form (Ranging from -4 (poor quality) to 4 (good quality)).

B.4. MAPPING OF THE PRIMARY STUDIES

ID	RQ1.5										RQ1.6										RQ1.7					RQ1.8		RQ1.9		Q.A.
	a	b	c	d	e	f	g	h	i	a	b	c	d	e	f	g	h	i	a	b	c	d	e	f	a	b	a	b		
(S49)	X			X		X				X																X		X		3
(S50)		X	X							X	X															X	X		X	-2
(S51)		X	X	X								X														X	X		X	0
(S52)	X			X		X						X														X	X		X	3
(S53)	X			X		X						X														X	X		X	2
(S54)		X	X	X																						X	X		X	-1
(S55)	X	X	X	X	X																					X	X		X	-1
(S56)	X	X	X	X	X																					X	X		X	1
(S57)	X		X	X	X																					X	X		X	-1
(S58)		X	X	X	X																					X	X		X	-2
(S59)		X	X	X	X																					X	X		X	-1
(S60)		X	X	X	X																					X	X		X	-1
(S61)	X			X																						X	X		X	2
(S62)		X	X	X	X																					X	X		X	1
(S63)	X		X	X	X																					X	X		X	1
(S64)	X		X	X	X																					X	X		X	1
(S65)		X	X	X	X																					X	X		X	2
(S66)		X	X	X	X																					X	X		X	-2
(S67)	X	X	X	X	X																					X	X		X	0
(S68)		X	X	X	X																					X	X		X	0
(S69)		X	X	X	X																					X	X		X	0
(S70)	X		X	X	X																					X	X		X	4
(S71)	X		X	X	X																					X	X		X	1
(S72)		X	X	X	X																					X	X		X	0
(S73)		X	X	X	X																					X	X		X	0
(S74)		X	X	X	X																					X	X		X	-1
(S75)		X	X	X	X																					X	X		X	0
(S76)		X	X	X	X																					X	X		X	1
(S77)	X		X	X	X																					X	X		X	0
(S78)	X		X	X	X																					X	X		X	1
(S79)	X		X	X	X																					X	X		X	0
(S80)		X	X	X	X																					X	X		X	1
(S81)		X	X	X	X																					X	X		X	1
(S82)		X	X	X	X																					X	X		X	1
(S83)	X		X	X	X																					X	X		X	-2
(S84)		X	X	X	X																					X	X		X	0
(S85)	X		X	X	X																					X	X		X	3
(S86)	X		X	X	X																					X	X		X	1
(S87)		X	X	X	X																					X	X		X	-2
(S88)	X		X	X	X																					X	X		X	-1

RQ1.5:(a) Manual; (b) Partially; (c) Automated; (d) Ad hoc; (e) Mathematical Formalism; (f) Structural (Re-engineering); (g) Semantics (Refactoring).
 RQ1.6:(a) Scoping; (b) Requirement; (c) Architecture; (d) Realization; (e) Test; (f) Requirement; (g) Architecture; (h) Realization; (i) Test.
 RQ1.7:(a) Not Evaluated; (b) Feasibility Study; (c) Controlled Experiments; (d) Case Studies; (e) Surveys; (f) Other.
 RQ1.8:(a) Automatic; (b) Manual.
 RQ1.9:(a) Academia; (b) Industry.
 Q.A.: Sum of the Values from the Quality Assessment Form (Ranging from -4 (poor quality) to 4 (good quality)).

ID	RQ1.5							RQ1.6							RQ1.7					RQ1.8		RQ1.9		Q.A.			
	a	b	c	d	e	f	g	a	b	c	d	e	f	g	h	i	a	b	c	d	e	f	a		b	a	b
(S89)	X		X	X		X	X	X	X	X							X	X					X	X			-1
(S90)	X		X	X		X	X	X	X								X	X					X	X			2
(S91)	X		X	X		X	X	X	X			X	X				X	X					X	X			-2
(S92)	X		X	X		X	X	X	X			X	X				X	X					X	X			0
(S93)	X		X	X		X	X	X	X			X	X				X	X					X	X			-1
(S94)	X		X	X		X	X	X	X			X	X				X	X					X	X			-1
(S95)	X		X	X		X	X	X	X			X	X				X	X					X	X			4
(S96)	X		X	X		X	X	X	X			X	X				X	X					X	X			3
(S97)	X		X	X		X	X	X	X			X	X				X	X					X	X			0
(S98)			X	X		X	X	X	X			X	X				X	X					X	X			-3
(S99)			X	X		X	X	X	X			X	X				X	X					X	X			0
(S100)	X		X	X		X	X	X	X			X	X				X	X					X	X			1
(S101)	X		X	X		X	X	X	X			X	X				X	X					X	X			2
(S102)	X		X	X		X	X	X	X			X	X				X	X					X	X			3
(S103)			X	X		X	X	X	X			X	X				X	X					X	X			3
(S104)	X		X	X		X	X	X	X			X	X				X	X					X	X			4
(S105)	X		X	X		X	X	X	X			X	X				X	X					X	X			3
(S106)	X		X	X		X	X	X	X			X	X				X	X					X	X			1
(S107)	X		X	X		X	X	X	X			X	X				X	X					X	X			-1
(S108)			X	X		X	X	X	X			X	X				X	X					X	X			1
(S109)			X	X		X	X	X	X			X	X				X	X					X	X			1
(S110)	X		X	X		X	X	X	X			X	X				X	X					X	X			0
(S111)			X	X		X	X	X	X			X	X				X	X					X	X			0
(S112)	X		X	X		X	X	X	X			X	X				X	X					X	X			0
(S113)	X		X	X		X	X	X	X			X	X				X	X					X	X			-1
(S114)	X		X	X		X	X	X	X			X	X				X	X					X	X			0
(S115)			X	X		X	X	X	X			X	X				X	X					X	X			0
(S116)	X		X	X		X	X	X	X			X	X				X	X					X	X			2
(S117)	X		X	X		X	X	X	X			X	X				X	X					X	X			-3
(S118)			X	X		X	X	X	X			X	X				X	X					X	X			-2
(S119)			X	X		X	X	X	X			X	X				X	X					X	X			0
(S120)			X	X		X	X	X	X			X	X				X	X					X	X			0
(S121)	X		X	X		X	X	X	X			X	X				X	X					X	X			-3
(S122)	X		X	X		X	X	X	X			X	X				X	X					X	X			-2
(S123)			X	X		X	X	X	X			X	X				X	X					X	X			-2
(S124)	X		X	X		X	X	X	X			X	X				X	X					X	X			-1
(S125)			X	X		X	X	X	X			X	X				X	X					X	X			0
(S126)	X		X	X		X	X	X	X			X	X				X	X					X	X			-1
(S127)	X		X	X		X	X	X	X			X	X				X	X					X	X			-1
(S128)	X		X	X		X	X	X	X			X	X				X	X					X	X			-1

RQ1.5:(a) Manual; (b) Partially; (c) Automated; (d) Ad hoc; (e) Mathematical Formalism; (f) Structural (Re-engineering); (g) Semantics (Refactoring).

RQ1.6:(a) Scoping; (b) Requirement; (c) Architecture; (d) Realization; (e) Test; (f) Requirement; (g) Architecture; (h) Realization; (i) Test.

RQ1.7:(a) Not Evaluated; (b) Feasibility Study; (c) Controlled Experiments; (d) Case Studies; (e) Surveys; (f) Other.

RQ1.8:(a) Automatic; (b) Manual.

RQ1.9:(a) Academia; (b) Industry.

Q.A: Sum of the Values from the Quality Assessment Form (Ranging from -4 (poor quality) to 4 (good quality)).

B.4. MAPPING OF THE PRIMARY STUDIES

ID	RQ1.5							RQ1.6							RQ1.7					RQ1.8		RQ1.9		Q.A.						
	a	b	c	d	e	f	g	a	b	c	d	e	f	g	h	i	a	b	c	d	e	f	a		b	a	b	a	b	
(S129)			X		X			X	X		X						X							X		X				-1
(S130)	X			X							X						X							X	X	X	X			-1
(S131)	X				X	X		X	X															X		X	X	X	X	-1
(S132)			X					X	X		X													X		X	X	X	X	-1
(S133)				X				X	X		X													X		X	X	X	X	-2
(S134)	X							X	X		X						X									X	X	X	X	1
(S135)					X			X	X															X		X	X	X	X	-1
(S136)				X				X	X		X													X		X	X	X	X	2
(S137)				X				X	X			X												X		X	X	X	X	-1
(S138)				X				X	X		X						X							X		X	X	X	X	3
(S139)			X					X	X								X	X									X	X	X	0
(S140)			X			X		X	X		X						X	X						X		X	X	X	X	-1
(S141)			X			X		X	X		X						X	X						X		X	X	X	X	-1
(S142)	X			X				X	X		X						X	X						X		X		X	X	0

RQ1.5:(a) Manual; (b) Partially; (c) Automated; (d) Ad hoc; (e) Mathematical Formalism; (f) Structural (Re-engineering); (g) Semantics (Refactoring).
 RQ1.6:(a) Scoping; (b) Requirement; (c) Architecture; (d) Realization; (e) Test; (f) Requirement; (g) Architecture; (h) Realization; (i) Test.
 RQ1.7:(a) Not Evaluated; (b) Feasibility Study; (c) Controlled Experiments; (d) Case Studies; (e) Surveys; (f) Other.
 RQ1.8:(a) Automatic; (b) Manual.
 RQ1.9:(a) Academia; (b) Industry.
 Q.A.: Sum of the Values from the Quality Assessment Form (Ranging from -4 (poor quality) to 4 (good quality)).



Feature-Driven Requirements Engineering (FeDRE) Approach

This appendix presents some relevant data from the empirical study performed to evaluate Feature-Driven Requirements Engineering (FeDRE) Approach, earlier discussed in [Chapter 5](#).

C.1 Survey Statements to Evaluate FeDRE, based on PEOU and PU variables

Variable	Statement
PEOU1	The FeDRE approach is simple and easy to follow
PEOU2	It is easy for me to follow the guidelines proposed by the FeDRE approach
PEOU3	The guidelines for specifying SPL functional requirements are easy to learn
PU1	I believe that the FeDRE approach would reduce the time required to specify SPL requirements
PU2	Overall, I found the FeDRE approach to be useful
PU3	I believe that the SPL requirements specifications obtained with the FeDRE approach are organized, clear, concise and non-ambiguous
PU4	I believe that the FeDRE approach has sufficient expressiveness to represent functional SPL requirements

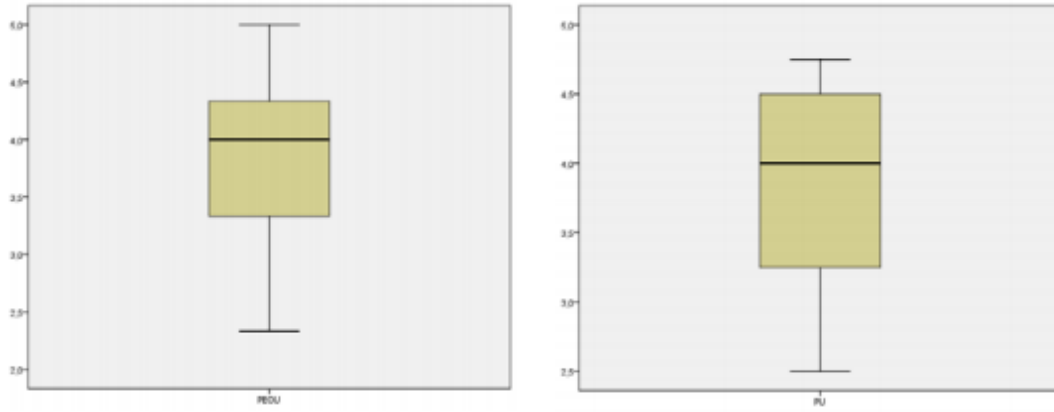
C.2 Identified Use Cases for each Feature

Feature Name	Associated to an Use Case or Alternative Use Case Scenario?	Use Case Name
Access_Control	Use Case	Create_User, Login, Show_Profile, Remember_Password, Send_E-mail
Web_Access_Control	Alternative scenario	Show_Profile
Mobile_Access_Control	Alternative scenario	Show_Profile
Import_Contact	Use Case	Retrieve_Contacts, Import_Contacts
Facebook_Import	Alternative scenario	Retrieve_Contacts
Twitter_Import	Alternative scenario	Retrieve_Contacts
Destination	Use Case	Send_Notification
SMS_Destination	Alternative scenario	Send_Notification
Twitter_Destination	Alternative scenario	Send_Notification
Facebook_Destination	Alternative scenario	Send_Notification
Email_Destination	Alternative scenario	Send_Notification
Contact	Use Case	Show_Contact, Delete_Contact, Update Contact
Add_Contact	Use Case	Add_Contact
Destination	Use Case	Send_Notification
Emergency_Numbers	Use Case	Create_Emergency_Number, Delete_Emergency_Number, Update_Emergency_Number

C.3 Subject's Responses for PEOU and PU

Subject	Session	PEOU1	PEOU2	PEOU3	PEOU	PU1	PU2	PU3	PU4	PU
Spain	1	4	3	4	3.66	1	5	5	5	4
	2	4	3	4	3.66	4	5	5	4	4.5
	3	4	2	2	2.66	4	4	5	5	4.5
	4	3	2	2	2.33	4	4	4	3	3.75
	5	2	3	4	3	2	3	2	3	2.5
	6	4	4	5	4.33	4	5	4	3	4
	7	4	2	4	3.33	3	3	3	2	2.75
	8	5	5	5	5	2	4	3	4	3.25
Brazil	1	4	4	4	4	5	5	5	4	4.75
	2	4	4	4	4	4	5	5	5	4.75
	3	4	4	4	4	3	4	2	2	2.75
	4	5	5	5	5	5	5	4	4	4.5
	5	4	4	5	4.33	3	5	5	4	4.25
	6	5	4	5	4.66	4	3	4	5	4

C.4 Box Plots for PEOU and PU Variables





Feature-Driven Requirements Engineering Evolution (FeDRE²) Approach

This appendix presents some relevant data from the empirical study performed to evaluate Feature-Driven Requirements Engineering Evolution (FeDRE²) Approach, earlier described in Chapter 6.

D.1 Survey Statements to Evaluate FeDRE², based on PEOU and PU variables

Variable	Statement
PU1	I believe that the evolved SPL requirements obtained with the FeDRE ² approach are disorganized, unclear, unconcise and ambiguous
PU2	I believe that the FeDRE ² approach has enough expressiveness to evolve functional SPL requirements
PU3	I believe that FeDRE ² approach would increase the time required to evolve SPL requirements
PU4	Overall, I found the FeDRE ² approach to be useful
PEOU1	It is difficult for me to follow the guidelines proposed by FeDRE ² approach
PEOU2	The guidelines for evolving SPL functional requirements are easy to learn
PEOU3	The identification of the evolution scenarios using FeDRE ² is complex and difficult to follow
PEOU4	Overall, the evolved requirements obtained by the FeDRE ² approach are easy to use
PEOU5	The SPL requirements evolution using FeDRE ² is simple and easy to follow
PEOU6	Updating the traceability matrix using FeDRE ² is simple and easy to follow

D.2 FeDRE² Background Form

GENERAL INFORMATION

1. Full Name *: _____

2. Degree (Bachelor, Master, PhD, PostDoc, ...)*: _____

3. Years since graduation (Mark only one oval.)*

< 1 year

>= 1 year and < 5 years

>= 5 years and < 10 years

> 10 years

TECHNICAL KNOWLEDGE

Select the option that best fits to your profile.

4. Regarding your Requirements Engineering background (Mark only one oval.)*

I have been involved in software development teams applying Requirements Engineering

I am a researcher working on topics related to Requirements Engineering

I know what Requirements Engineering is but I have never participated in a software project applying Requirements Engineering

I have never heard about Requirements Engineering

5. How many years of experience do you have in Requirements Engineering? (Mark only one oval.)*

< 1 year

>= 1 year and < 5 years

>= 5 years and < 10 years

> 10 years

6. Have you applied Requirements Engineering in building software? (Mark only one oval.)*

Yes, but only in the research domain

Yes, but only in the industry domain

Yes, both research and industry domain

No

7. Regarding your Software Evolution background (Mark only one oval.)*

I have been involved in software development teams dealing with Software Evolution

I am a researcher working on topics related to Software Evolution

I know what Software Evolution is but I have never participated in a software project

dealing with Software Evolution

I have never heard about Software Evolution

8. How many years of experience do you have in Software Evolution? (Mark only one oval.)*

< 1 year

>= 1 year and < 5 years

>= 5 years and < 10 years

> 10 years

9. Have you ever dealt with Software Evolution? (Mark only one oval.)*

Yes, but only in the research domain

Yes, but only in the industry domain

Yes, both research and industry domain

No

10. Regarding your SPL background (Mark only one oval.)*

I have been involved in software development teams applying the Software Product Line approach

I am a researcher working on topics related to Software Product Line Development

I know what Product Lines are but I have never participated in a software project applying SPL development

I have never heard about Software Product Lines

11. How many years of experience do you have in SPL? (Mark only one oval.)*

< 1 year

>= 1 year and < 5 years

>= 5 years and < 10 years

> 10 years

12. Have you applied the SPL approach in building software? (Mark only one oval.)*

Yes, but only in the research domain

Yes, but only in the industry domain

Yes, both research and industry domain

No

13. A Software Product Line (SPL) is a: (Mark only one oval.)*

Set of products built from a platform of components

Set of software intensive systems sharing a common managed set of features developed from a common set of core assets in a prescribed way

I am not sure about that

14. What is a feature? (Mark only one oval.)*

A feature is a user-visible characteristic of a system or software product and which are usually organized in feature models

A feature is everything that can vary through an SPL

I am not sure about that

15. In SPL Development ... (Mark only one oval.)*

... requirements should be built from the scratch for every single product

... requirements concepts are not applicable

... there are: i) the Product Line Requirements with should include the variation mechanisms to cover all the possible products within the SPL and; ii) the Product Requirements that are derived from the Product Line Requirements by exercising these variation mechanisms in order to achieve a specific product

16. In SPL development... (Mark only one oval.)*

... an evolution is applied to a single product

... it is necessary to evolve a product line in accordance with changes to the environment, the market, and/or technology

... the concept of evolution is not applicable to SPL development

*** Required**

D.3 FeDRE² Survey

1. Full Name *: _____

Please read carefully each statement before answering.

2. I believe that the evolved SPL requirements obtained with the FeDRE² approach are disorganized, unclear, unconcise and ambiguous (Mark only one oval)*

	1	2	3	4	5	
Totally disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Totally agree

3. I believe that the FeDRE² approach has enough expressiveness to evolve functional SPL requirements (Mark only one oval)*

	1	2	3	4	5	
Totally disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Totally agree

4. It is difficult for me to follow the guidelines proposed by FeDRE² approach (Mark only one oval)*

	1	2	3	4	5	
Totally disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Totally agree

5. The guidelines for evolving SPL functional requirements are easy to learn (Mark only one oval)*

	1	2	3	4	5	
Totally disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Totally agree

6. I believe that FeDRE² approach would increase the time required to evolve SPL requirements (Mark only one oval)*

	1	2	3	4	5	
Totally disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Totally agree

7. Overall, I found the FeDRE² approach to be useful (Mark only one oval)*

	1	2	3	4	5	
Totally disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Totally agree

8. The identification of the evolution scenarios using FeDRE² is complex and difficult to follow (Mark only one oval)*

	1	2	3	4	5	
Totally disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Totally agree

9. The SPL requirements evolution using FeDRE² is simple and easy to follow (Mark only one oval)*

	1	2	3	4	5	
Totally disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Totally agree

10. Updating the traceability matrix using FeDRE² is simple and easy to follow (Mark only one oval)*

	1	2	3	4	5	
Totally disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Totally agree

11. Overall, the evolved requirements obtained by the FeDRE² approach are easy to use (Mark only one oval)*

	1	2	3	4	5	
Totally disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Totally agree

12. Do you have any suggestion on how to make this requirement evolution approach more easy to use?

13. What are the reasons that will make usable or not this approach in the future?

14. Please write any other comment or suggestion related to requirements evolution approach in the space below.

D.4 Subject's Responses for PEOU and PU

Subject	PU1	PU2	PU3	PU4	PU Mean	PEOU1	PEOU2	PEOU3	PEOU4	PEOU5	PEOU6	PEOU Mean
Brazil	1	4	2	5	3.25	3	3	5	4	4	5	4
	2	5	4	5	4.75	5	5	5	5	5	5	5
	3	5	5	5	5	5	5	5	5	5	5	5
	4	5	4	5	4.5	5	5	5	5	5	5	5
	5	5	5	5	4.25	5	4	5	5	5	5	4.83
	6	5	5	5	5	4	5	4	5	4	5	4.5
	7	4	4	3	4	3.75	5	5	4	5	4	4.67
	8	5	4	4	5	4.5	4	4	5	4	4	4.17
	9	4	5	5	5	4.75	4	4	4	4	4	4.17
Spain	10	5	4	3	3.75	4	4	4	4	4	4	4
	11	2	3	4	3	3	2	2	3	3	4	2.83
	12	2	4	3	3	2	2	4	4	4	5	3.5
	13	1	2	3	1	1.75	1	1	1	1	4	1.5
	14	5	5	5	5	5	4	4	5	5	5	4.5
	15	4	5	4	5	4.5	5	5	4	4	5	4.5
	16	5	4	4	5	4.5	2	4	4	4	4	3.83