



Pós-Graduação em Ciência da Computação

# “RiPLE-EM: A Process to Manage Evolution in Software Product Lines”

By

**Thiago Henrique Burgos de Oliveira**

M.Sc. Dissertation



Universidade Federal de Pernambuco  
posgraduacao@cin.ufpe.br  
[www.cin.ufpe.br/~posgraduacao](http://www.cin.ufpe.br/~posgraduacao)

RECIFE, AUGUST, 2009



Universidade Federal de Pernambuco  
Centro de Informática  
Pós-graduação em Ciência da Computação

Thiago Henrique Burgos de Oliveira

## **“RiPLE-EM: A Process to Manage Evolution in Software Product Lines”**

*Trabalho apresentado ao Programa de Pós-graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.*

*A M.Sc. Dissertation presented to the Federal University of Pernambuco in partial fulfillment of the requirements for the degree of M.Sc. in Computer Science.*

Advisor: *Silvio Romero de Lemos Meira*  
Co-Advisor: *Eduardo Santana de Almeida*

RECIFE, AUGUST, 2009

**Oliveira, Thiago Henrique Burgos de**  
**RiPLE-EM: a process to manage evolution in**  
**software product lines / Thiago Henrique Burgos de**  
**Oliveira. - Recife: O Autor, 2009.**  
**xv, 134 p. : fig., tab.**

**Dissertação (mestrado) – Universidade Federal de**  
**Pernambuco. Cln. Ciência da Computação, 2009.**

**Inclui bibliografia e apêndice.**

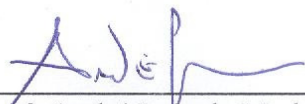
**1. Engenharia de software. I. Título.**

**005.1**

**CDD (22. ed.)**

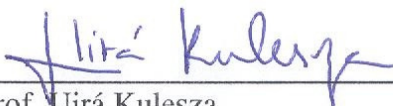
**MEI2009- 140**

Dissertação de Mestrado apresentada por **Thiago Henrique Burgos de Oliveira** à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título "**RiPLE-EM: A Process to Manage Evolution in Software Product Lines**", orientada pelo **Prof. Silvio Romero de Lemos Meira** e aprovada pela Banca Examinadora formada pelos professores:



---

Prof. André Luis de Medeiros Santos  
Centro de Informática / UFPE



---

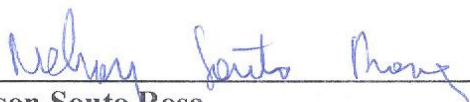
Prof. Uirá Kulesza  
Depto. De Informática e Matemática Aplicada / UFRN



---

Prof. Silvio Romero de Lemos Meira  
Centro de Informática / UFPE

Visto e permitida a impressão.  
Recife, 25 de agosto de 2009.



---

**Prof. Nelson Souto Rosa**

Vice-Coordenador da Pós-Graduação em Ciência da Computação do  
Centro de Informática da Universidade Federal de Pernambuco.

*This work is dedicated to my beloved wife, my parents, and  
my grandparents.*

# Acknowledgements

Lots of people were directly or indirectly involved in this work, and my intention is to thank all of them, however, it is just not possible to remember all of them now. My excuses for the ones I forgot, it definitely does not mean you were not important.

Initially, I would like to thank God for the opportunity and for all the blessings from the heaven above. Without the presence of God, nothing in my life would be as it is. I thank God for the protection, and guidance during this journey, and all my life.

Next, I would like to thank my family. Starting with my beloved wife, Debora, who supported me at all times, giving me strength to finish my work. God have blessed me so much with her fellowship and my words are not enough to express my gratitude to her. Next, my mother, my father and my grand parents who always invested on me, and are responsible for the things I have achieved. To my brother, my love, gratitude and example of roads to course.

I would like to thank C.E.S.A.R for all the vital support in my studies, the flexibility to attend classes and the flexibility to work from a different country, while I was focusing on my studies at Germany.

I would like to thank all the RiSE members for their kindness and patience, specially Eduardo Almeida, my co-advisor, who started teaching me the paths of academic research, and granted me an amazing experience abroad, at Germany.

My Gratitude to the Fraunhofer Institute for Software Experimentation (IESE), where I spent 3 months, working on my dissertation proposal. Some key people from the Fraunhofer IESE were essential in this partnership. Dirk Muthig, for accepting me as a visiting researcher, Michalis Anastasopoulos for the mentoring and attention not only while the time I was there but also after I left, and Vander Alves for the friendship.

Last, but not least, I would like thank all my friends and relatives who have always supported and prayed for me.

*If you can dream it, you can do it.*  
— WALT DISNEY (1901-1966)

Reuso de software é um aspecto chave para organizações interessadas em obter melhorias de produtividade, qualidade e redução de custos. Linhas de Produto de Software é uma abordagem de reuso de software que provou seus benefícios em diferentes contextos industriais (Weiss *et al.*, 2006). Em termos de evolução, uma linha de produtos é um conjunto em contínua evolução, e por isso, sua evolução precisa ser gerenciada para que se alcance os benefícios dessa abordagem.

O fato de um *core asset* ser compartilhado entre produtos, e todas as mudanças neste *core asset* poder ter efeito sobre diversos produtos (McGregor, 2003), aliado ao fato que em linhas de produto de software é preciso lidar com evolução no tempo (versões) e também evolução no espaço (variabilidade) (Krueger, 2002), faz com que o gerenciamento da evolução (mudanças) em linhas de produto de software seja mais complexo e mais desafiador do que o desenvolvimento tradicional de sistemas únicos (Pussinen, 2002). Portanto, a evolução dos *core assets* e também dos produtos precisa ser bem gerenciada para minimizar os problemas causados por ela.

Este desafio envolve diferentes soluções, como questões técnicas, gerenciais e processuais. Desta forma, o foco desta dissertação está nos problemas ligados ao processo de gerenciamento evolução em linhas de produto de software.

Neste contexto, este trabalho apresenta o RiPLE-EM, que é um processo para gerenciamento da evolução. Este processo é uma forma sistemática de guiar e gerenciar a evolução de cada *core asset* e cada produto, englobando atividades de gerenciamento de mudanças, *builds*, e entregas.

Esta dissertação também apresenta a validação inicial do RiPLE-EM, seguindo guias bem estabelecidos de experimentação de software (Wohlin *et al.*, 2000), e de acordo com os dados coletados e analisados na experimentação, RiPLE-EM mostra indicações de que seja um processo viável para o gerenciamento da evolução em linhas de produto de software.

**Palavras-chave:** Software Product Lines, Evolution, Evolution Management, Release Management, Build Management, Change Management



# Abstract

Software reuse is a key aspect for organizations interested in achieving improvements in productivity, quality and costs reduction. Software product lines, as a software reuse approach, have proven its benefits in different industrial environments (Weiss *et al.*, 2006). In terms of evolution, a product line is a continuously evolving organism, and for that, evolution should be managed properly to achieve all benefits of this approach.

The fact that a core asset is shared among products, and every change to this asset can have effects on several products (McGregor, 2003), combined with the fact that in the SPL we have to deal with evolution in time (versions) and space (variability) (Krueger, 2002), make evolution management more challenging than in traditional single software development (Pussinen, 2002). Thus the evolution of each core asset and product need to be well managed to minimize problems like this.

Thus, those challenges involve different solution spaces, such as technical issues, managerial and processes issues. Nevertheless, this dissertation focus is on the process issues of evolution management of software product lines.

In this context, this dissertation presents the RiPLE-EM process to evolution management. The process is a systematic way to guide and manage the evolution of every asset and product in a product line context, handling change management, build management and release management activities.

This dissertation also presented the initial validation of RiPLE-EM process, following well established guidelines to software experimentation (Wohlin *et al.*, 2000), and according to the data collected and analyzed in the experimental study, RiPLE-EM presents indications that the process can be viable.

**Keywords:** Software Product Lines, Evolution, Evolution Management, Release Management, Build Management, Change Management

# Contents

<b>Acknowledgments</b>	<b>v</b>
<b>Resumo</b>	<b>vii</b>
<b>Abstract</b>	<b>viii</b>
<b>Table of Contents</b>	<b>xii</b>
<b>List of Figures</b>	<b>xiv</b>
<b>List of Tables</b>	<b>xv</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	2
1.2 Problem Statement . . . . .	2
1.3 Overview of the Solution . . . . .	3
1.3.1 Context . . . . .	3
1.3.2 Proposal Outline . . . . .	4
1.4 Out of Scope . . . . .	5
1.5 Related Work . . . . .	6
1.6 Dissertation Structure . . . . .	6
<b>2 SPL and Evolution</b>	<b>8</b>
2.1 SPL . . . . .	8
2.1.1 Essential Activities . . . . .	9
2.1.2 Variability Management . . . . .	11
2.1.3 Adoption Strategies . . . . .	13
2.2 Software Evolution . . . . .	14
2.2.1 Evolution Laws . . . . .	15
2.2.2 Evolution Dimensions . . . . .	16
2.3 SPL Evolution . . . . .	17
2.3.1 Forces for Change . . . . .	17
2.3.2 Evolution Propagation . . . . .	18
2.3.3 SPL Evolution . . . . .	19
2.3.4 Challenges . . . . .	20

---

2.4	Chapter Summary . . . . .	20
<b>3</b>	<b>Systematic Review</b>	<b>22</b>
3.1	Systematic Literature Reviews . . . . .	23
3.2	Planning . . . . .	23
3.2.1	Question Structure . . . . .	23
3.2.2	Research Questions . . . . .	24
3.2.3	Threats to Validity . . . . .	27
3.2.4	Inclusion and Exclusion Criteria . . . . .	28
3.3	Conduction . . . . .	28
3.3.1	Search Strategy . . . . .	29
3.3.2	Query Strings . . . . .	29
3.3.3	Data Sources . . . . .	30
3.3.4	Studies Selection . . . . .	30
3.3.5	Data Extraction . . . . .	32
3.3.6	Data Analysis and Synthesis . . . . .	33
3.4	Reporting . . . . .	33
3.4.1	Configuration Identification . . . . .	33
3.4.2	Multi-Level Instantiation . . . . .	33
3.4.3	Release Management . . . . .	34
3.4.4	Change Management . . . . .	36
	Variability Evolution . . . . .	38
	Change Propagation . . . . .	38
3.4.5	Build Management . . . . .	39
3.4.6	Questions Summary . . . . .	39
3.5	Lessons Learned . . . . .	40
3.6	Related Work . . . . .	41
3.7	Chapter Summary . . . . .	41
<b>4</b>	<b>RiPLE-EM</b>	<b>42</b>
4.1	RiPLE-EM Overview . . . . .	42
4.2	RiPLE-EM Disciplines . . . . .	44
4.3	RiPLE-EM Usage Scenarios . . . . .	45
4.4	RiPLE-EM CAD x PD . . . . .	46
4.4.1	RiPLE-EM CAD X PD Communication . . . . .	47
4.5	RiPLE-EM Support Tools . . . . .	48

---

---

4.6	RiPLE-EM Related Roles . . . . .	49
4.7	RiPLE-EM Activities Summary . . . . .	50
4.8	RiPLE-EM Work Products . . . . .	52
4.9	Eclipse Process Framework . . . . .	53
4.9.1	Method Content and Processes . . . . .	53
4.9.2	Benefits . . . . .	54
4.10	Chapter Summary . . . . .	54
<b>5</b>	<b>RiPLE-EM CAD</b>	<b>56</b>
5.1	RiPLE-EM CAD Flow . . . . .	56
5.2	RiPLE-EM CAD - Release Planning . . . . .	57
5.3	RiPLE-EM CAD - Change Management . . . . .	60
5.4	RiPLE-EM CAD - Build Management . . . . .	69
5.5	RiPLE-EM CAD - Release Execution . . . . .	73
5.6	Chapter Summary . . . . .	76
<b>6</b>	<b>RiPLE-EM PD</b>	<b>77</b>
6.1	RiPLE-EM PD Flow . . . . .	77
6.2	RiPLE-EM PD - Release Planning . . . . .	78
6.3	RiPLE-EM PD - Change Management . . . . .	81
6.4	RiPLE-EM PD - Build Management . . . . .	86
6.5	RiPLE-EM PD - Release Execution . . . . .	88
6.6	Chapter Summary . . . . .	90
<b>7</b>	<b>The Experimental Study</b>	<b>92</b>
7.1	The Definition . . . . .	93
7.1.1	Goal . . . . .	93
7.1.2	Questions . . . . .	94
7.1.3	Metrics . . . . .	94
	Definition Summary . . . . .	95
7.2	The Planning . . . . .	96
7.2.1	Context Selection . . . . .	96
7.2.2	Hypothesis . . . . .	96
7.2.3	Variables and Treatments . . . . .	97
7.2.4	Subjects . . . . .	97
7.2.5	Experiment Design . . . . .	98

---

---

7.2.6	Instrumentation . . . . .	98
7.2.7	Validity Evaluation . . . . .	99
7.3	The Operation . . . . .	102
7.3.1	Preparation . . . . .	102
7.3.2	Execution . . . . .	102
7.3.3	Data Validation . . . . .	103
7.4	Analysis and interpretation . . . . .	103
7.4.1	Effort to Apply the Process . . . . .	103
7.4.2	Difficulties to understand/apply the process . . . . .	104
7.4.3	Activities, Roles and Artifacts Missing . . . . .	105
7.4.4	Uncompleted Propagation Requests . . . . .	106
7.4.5	Subjects Satisfaction . . . . .	106
7.5	Lessons Learned . . . . .	107
7.6	Chapter Summary . . . . .	108
<b>8</b>	<b>Conclusions</b>	<b>109</b>
8.1	Research Contributions . . . . .	110
8.2	Related Work . . . . .	111
8.3	Future Work . . . . .	111
8.4	Academic Contributions . . . . .	113
8.5	Joint Contributions . . . . .	113
8.6	Concluding Remarks . . . . .	116
	<b>Bibliography</b>	<b>118</b>
	<b>Appendices</b>	<b>125</b>
<b>A</b>	<b>Experiment Questionnaires</b>	<b>126</b>
A.1	QT1 . . . . .	126
A.2	QT2 . . . . .	127
<b>B</b>	<b>Checklists and Templates</b>	<b>128</b>
B.1	CR Checklist . . . . .	128
B.1.1	Checks . . . . .	129
B.2	PR Checklist . . . . .	130
B.2.1	Checks . . . . .	130
B.3	CCB Template . . . . .	130

---

---

B.3.1	Template . . . . .	130
<b>C</b>	<b>Systematic Review Sources</b>	<b>132</b>
C.1	Journals . . . . .	132
C.2	Conferences . . . . .	133
C.3	Web-Search . . . . .	134

# List of Figures

1.1	RiSE Labs Influences . . . . .	3
1.2	RiSE Labs Projects . . . . .	5
1.3	Roadmap to this dissertation . . . . .	7
2.1	Software Product Lines Essential Activities (Clements and Northrop, 2002)	10
2.2	SPL Core Asset Development (Clements and Northrop, 2002) . . . . .	11
2.3	SPL Production Plan (Clements and Northrop, 2002) . . . . .	12
2.4	SPL Product Development (Clements and Northrop, 2002) . . . . .	13
4.1	RiPLE-EM Main flow . . . . .	43
4.2	RiPLE-EM Big Picture . . . . .	43
4.3	RiPLE-EM usage example . . . . .	45
4.4	RiPLE-EM usage example . . . . .	46
4.5	RiPLE-EM usage example . . . . .	46
4.6	RiPLE-EM Change Propagation Request . . . . .	48
5.1	RiPLE-EM CAD Macro Flow . . . . .	57
5.2	RiPLE-EM CAD: Release Planning Flow . . . . .	58
5.3	RiPLE-EM CAD: Change Management Flow . . . . .	61
5.4	RiPLE-EM CAD: Process Request Flow . . . . .	63
5.5	RiPLE-EM CAD: Process Change Request Flow . . . . .	64
5.6	RiPLE-EM CAD:Process Propagation Request Flow . . . . .	67
5.7	RiPLE-EM CAD: Build Management Flow . . . . .	70
5.8	RiPLE-EM CAD: Release Execution Flow . . . . .	74
6.1	RiPLE-EM PD Macro Flow . . . . .	78
6.2	RiPLE-EM PD: Release Planning Flow . . . . .	79
6.3	RiPLE-EM PD: Change Management Flow . . . . .	82
6.4	RiPLE-EM PD: Change Management - Process Requests . . . . .	83
6.5	RiPLE-EM PD: Change Management - Process Change Request . . . . .	84
6.6	RiPLE-EM PD: Change Management - Process Propagation Request . . . . .	85
6.7	RiPLE-EM PD: Build Management Flow . . . . .	87
6.8	RiPLE-EM PD: Release Execution Flow . . . . .	89
7.1	Effort Pie Chart . . . . .	104

---

7.2	Difficulties Distribution Chart . . . . .	105
7.3	Subjects satisfaction levels . . . . .	106
8.1	Solution Layers . . . . .	114
8.2	RiPLE-EM and CL integration . . . . .	115



# List of Tables

2.1	Differences between single-system development and Software product lines	20
3.1	Common issues raised x Recommended change management practice .	36
3.2	Summary table of all sub-questions and approaches match . . . . .	39
4.1	RiPLE-EM CAD and RiPLE-EM PD Main Differences . . . . .	47
4.2	RiPLE-EM Change Management Activities Summary . . . . .	51
4.3	RiPLE-EM Build Management Activities Summary . . . . .	51
4.4	RiPLE-EM Release Management Activities Summary . . . . .	52
7.1	Subject's Profile in the Experimental Study . . . . .	103
7.2	Effort to use the process . . . . .	104
7.3	Effort to use the process . . . . .	105
7.4	Hypothesis Rejection Summary . . . . .	108

“With the possible exception of the equator, everything begins somewhere.”

C. S. Lewis



# Introduction

In order to minimize costs and time-to-market, and maximize quality and productivity, *software reuse* is an important aspect. These goals are usually leveraged by the tendency of systems to get bigger and complex. In addition to that, the application of *ad-hoc* reuse (which is an opportunistic reuse, not systematic, and generally restricted to source code) also leverage the goals cited, by inserting risks that can compromise future initiatives in the systematic reuse direction (Almeida, 2007).

In order to achieve these benefits (high productivity, high quality, low cost and less time-to-market), one of the approaches with significantly and increasing success are software product lines (Weiss *et al.*, 2006).

Although software product lines have proven its applicability and benefits in a broad range of domains (Weiss *et al.*, 2006), some specific topics regarding it have not been widely explored, leaving an open gap for researchers and practioners. The evolution management of assets and product inside the product line is an example of a filed that has not been developed in academy. Thus, the focus of this dissertation is to study the state-of-the-art in evolution management techniques for software product lines and provide a systematic process to manage evolution of assets and products inside software product lines, in order to maximize the benefits of systematic reuse.

This Chapter contextualizes the focus of this dissertation and starts by presenting its motivation in Section 1.1 and a clear definition of the problem in Section 1.2. A brief overview of the proposed solution is presented in Section 1.3, while Section 1.4 describes some related aspects that are not directly addressed by this work and Section 1.5 describes some work related to this dissertation. Section 1.6 outlines the remainder structure of this dissertation.

## 1.1 Motivation

Although the notion of software product lines have been studied by several authors (Clements and Northrop, 2002; Atkinson *et al.*, 2002; Pohl *et al.*, 2005a; Bayer *et al.*, 1999; Weiss and Lai, 1999), the primary efforts are targeted at the conversion towards (Pohl *et al.*, 2005b) and the initiation of a software product line (Clements and Northrop, 2002) and, consequently, evolution is considerably less studied. On the other hand, a software product line and its assets, as any other piece of software, evolves over time. The evolution of a product line and its assets is driven by inclusions and changes in the requirements set of the products inside the product line. These changes are originated from a number of different sources, such as the customers using the products, future needs predicted by the company, bug fixes and the introduction of new products in the product line.

This evolution management in the context of software product line is, most times, a very complex activity for different reasons, such as the assets shared among products, variability management, changes propagation, among others. Some issues can be identified in the software product lines evolution management field such as technical question (repositories, CASE tools) Anastasopoulos *et al.* (2009) and managerial and process issues related to the coordination of evolution management activities.

The fact that a core asset is shared among products, and every change to this asset can have effects on several products (McGregor, 2003), combined with the fact that in the SPL we have to deal with evolution in time (versions) and space (variability) (Krueger, 2002), make SPL evolution management much more complex and challenging. Thus the evolution of each core asset and product need to be well managed to minimize problems like this.

In this work, we try to address evolution management, in the context of product lines, by defining a process with activities necessary to manage evolution properly, in terms of change, build and release management. Both software product lines and evolution management in the context of software product lines are further discussed in Chapter 2.

## 1.2 Problem Statement

Motivated by the scenario presented in the previous Section, the goal of the work described in this dissertation can be stated as:

**This work defines a process to manage evolution of assets and products, in the**

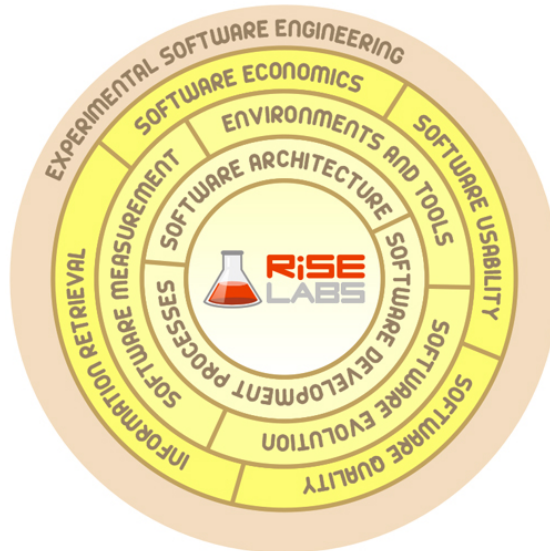
context of software product lines, by defining workflows, roles and artifacts for managing changes, builds and releases, aiming at making the evolution of assets and products a controlled and systematic task, in order to achieve systematic reuse in an effective way.

## 1.3 Overview of the Proposed Solution

In order to accomplish the goal of this dissertation, RiPLE-EM (RiSE Product Line Engineering - Evolution Management) process is proposed. This Section presents the context where it is inserted and outlines the proposed solution.

### 1.3.1 Context

This dissertation is part of the RiSE Labs ([Almeida et al., 2004](#)), formerly called RiSE Project, whose goal is to develop a robust framework for software reuse in order to enable the adoption of a reuse program. Thus, it is influenced by several areas, such as software measurement, architecture, quality, environments, tools, and so on, in order to achieve its goal. The influence areas are depicted in Figure 1.1. Based on these areas, the RiSE Labs is divided in several projects, as shown in Figure 1.2. As it can be seen in the Figure, this framework embraces several different projects related to software reuse. They are:



**Figure 1.1** RiSE Labs Influences

- **RiSE Framework:** Involves reuse processes (Almeida *et al.*, 2005), component certification (Alvaro *et al.*, 2006) and reuse adoption process (Garcia *et al.*, 2008).
- **RiSE Tools:** Research focused on software reuse tools, such as the Admire Environment (Mascena *et al.*, 2006), the Basic Asset Retrieval Tool (B.A.R.T) (Santos *et al.*, 2006), which was enhanced with folksonomy mechanisms (Vanderlei *et al.*, 2007), semantic layer (Duraõ, 2008), facets Mendes (2008) and data mining (Martins *et al.*, 2008), the Legacy InFormation retrieval Tool (LIFT) (dos Santos Brito *et al.*, 2008), a Tool for Domain Analysis (ToolDAY) (Lisboa, 2008), and the Component Repository (Burégio, 2006).
- **RiPLE:** Stands for *RiSE Product Lines Engineering* and aims at developing a methodology for software product lines. It is composed of disciplines related to scoping, requirements, design, implementation, testing and evolution.
- **SOPLE:** Development of a methodology for Software Product Lines based on services, following the same structure of RiPLE.
- **MATRIX:** Investigates the area of measurement in reuse and its impact in quality and productivity, based on strong experimentation.
- **BTT:** Research focused on tools for detection of duplicated change requests (Cavalcanti *et al.*, 2008), and its impact on reuse.
- **Exploratory Research:** Investigates new research directions in software engineering and its impact on reuse, based on packages of empirical studies.
- **CX-Ray:** Focused on understanding the C.E.S.A.R<sup>1</sup>, its processes and practices in software development.

This dissertation is part of the RiPLE project and its goal is to support the evolution management of assets and products in a software product line.

#### 1.3.2 Proposal Outline

The goal of this dissertation is to manage the evolution of assets and products in the software product line, by defining a systematic process composed by three main activities:

---

<sup>1</sup>C.E.S.A.R - Recife Center for Advanced Studies and Systems - <http://www.cesar.org.br>



**Figure 1.2** RiSE Labs Projects

**Change Management, Build Management and Release Management**, integrated in a macroflow that guides all evolution management activities.

This proposed process does not exclude existing evolution management techniques, patterns and tools (such as software configuration management systems), but comes to complement traditional evolution management with three activities focused on the software product lines context.

## 1.4 Out of Scope

As the proposed process is part of a broader context (RiPLE), a set of related aspects will be left out of its scope. Thus, the following issues are not directly addressed by this work:

- **Pro-active evolution.** There are some studies related to pro-active software evolution, related to predicting changes and preparing the product line to these changes. In the proposed process, the main focus is on reactive evolution, starting from the moment the need for change arises (be it a predicted change or not).
- **Guidelines to modify different types of assets.** The proposed process contains activity flows to guide activities in the change management area, however, it does not provide specific guidelines on how to change each type of asset (e.g. requirements, architecture, source code, feature models and etc). The modification of the assets should be done according with the development technique/methodology

being followed. The proposed process only sets the path from the change necessity arrival until the proper change of the artifact. Thus, the change itself (how it is performed) is not handled.

- **Evolution Metrics.** Measurement activities are essential in any engineering process. Both measurement activities inside the process and metrics to be used outside the process (to formally evaluate) could be incorporated to the process.

## 1.5 Related Work

Part of this study was conducted inside the Fraunhofer Software Engineering Institute (Fraunhofer IESE)<sup>2</sup> along with SPL specialists, thus this work had a great influence from the Fraunhofer IESE.

Specifically, this work had a great influence from Michalis Anastasopoulos, who works at the Fraunhofer and researches a research in a similar area. Michalis' research area is in a SPL tool to bridge the gap of the version control tools and the SPL concept. The interaction between this dissertation and Michalis' work, is better described in [Anastasopoulos \*et al.\* \(2009\)](#).

## 1.6 Dissertation Structure

The remainder of this dissertation is organized as follows:

- **Chapter 2** discusses software product lines basic concepts and activities, as well as software evolution. The relation between software product lines and software evolution is also discussed.
- **Chapter 3** presents a systematic review on evolution management existing approaches in the context of software product lines with the objective of mapping them to better understand the state-of-the-art in this field and to serve as basis for the proposed process.
- **Chapter 4** describes the proposed process to manage evolution in SPL, the roles associated, the disciplines involved and the key concepts of the process. It presents an overview picture of RiPLE-EM.

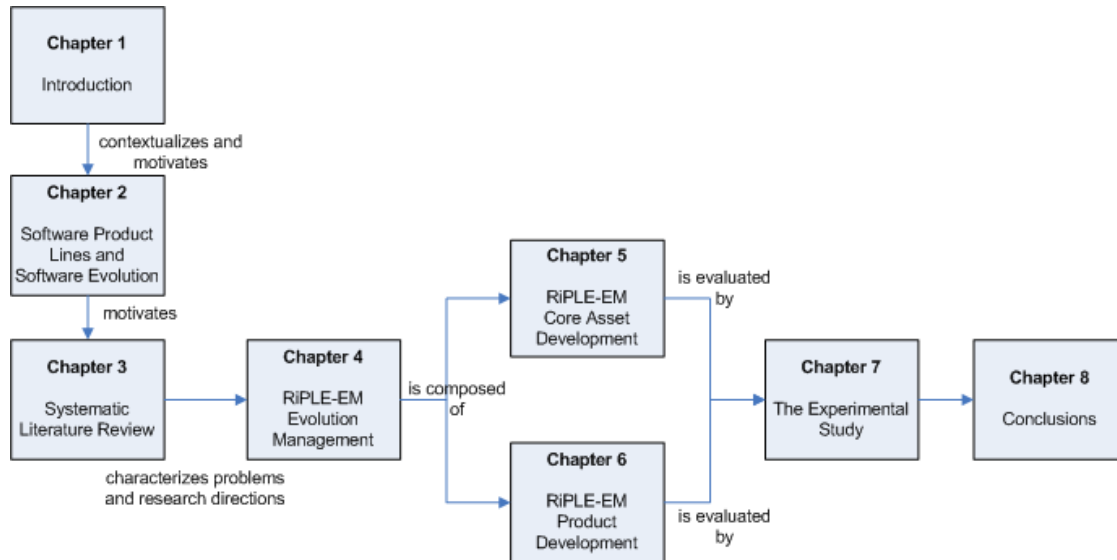
---

<sup>2</sup>Fraunhofer IESE - <http://www.iese.fraunhofer.de/de/>

---

- **Chapter 5** describes the proposed process to manage evolution in SPL for the Core Asset Development level, with and the existing activities, roles and steps.
- **Chapter 6** describes the proposed process to manage evolution in SPL for the Product Development, with the existing activities, roles and steps.
- **Chapter 7** presents the definition, planning, operation, analysis and interpretation and packaging of the experimental study which evaluates the use of the proposed process.
- **Chapter 8** concludes this dissertation by summarizing the findings and proposing future enhancements to the solution, along with the concluding remarks.
- **Appendix A** shows the questionnaires applied in the experimental study.
- **Appendix B** presents the checklists and templates proposed by this dissertation, to support RiPLE-EM.
- **Appendix C** presents the systematic review sources, where the primary studies were seached for.

Figure 1.3 represents the roadmap to the dissertation. The references and appendix were omitted from this roadmap to improve the legibility.



**Figure 1.3** Roadmap to this dissertation



*“What saves a man is to take a step. Then another step.”*

C. S. Lewis

# 2

## Software Product Lines and Software Evolution

In early 1967 there was an increasing importance and impact of software systems in many activities of society. In addition, as a result of the problems faced in software manufacturing, there was a general belief that available techniques to build software should be less ad hoc, and instead, it should be based on theoretical foundations, as an 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 discussed activities of software engineering, both maintenance (evolution) and software mass customization (principle for software product lines) were considered. Those two software development activities (which turned into formal software engineering fields later on) are discussed in this Chapter.

Thus, this Chapter introduces the concepts of software product lines in Section 2.1, and the concepts surrounding software evolution in Section 2.2. The evolution management in the context of software product lines is discussed in Section 2.3 and the Chapter summary is described in Section 2.4.

### 2.1 Software Product Lines

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

In the domain of automobiles 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.*, 2005b), however, not all people want the same kind of car for any purpose. Thus, industry was confronted with the rising interest for individualized products, which was the beginning of mass customization. However, mass customization had two distinct faces. For the customer, mass customization means the ability to have an individualized product. For the company, however, it means technological investments, which leads to higher product's prices and/or lower profit margins for the company (Pohl *et al.*, 2005b).

Thus, 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. In this ways, the use of platforms for different products can lead to reduction in the production cost for a particular kind of product. The systematic combination of mass customization and common platforms is the key for product lines.

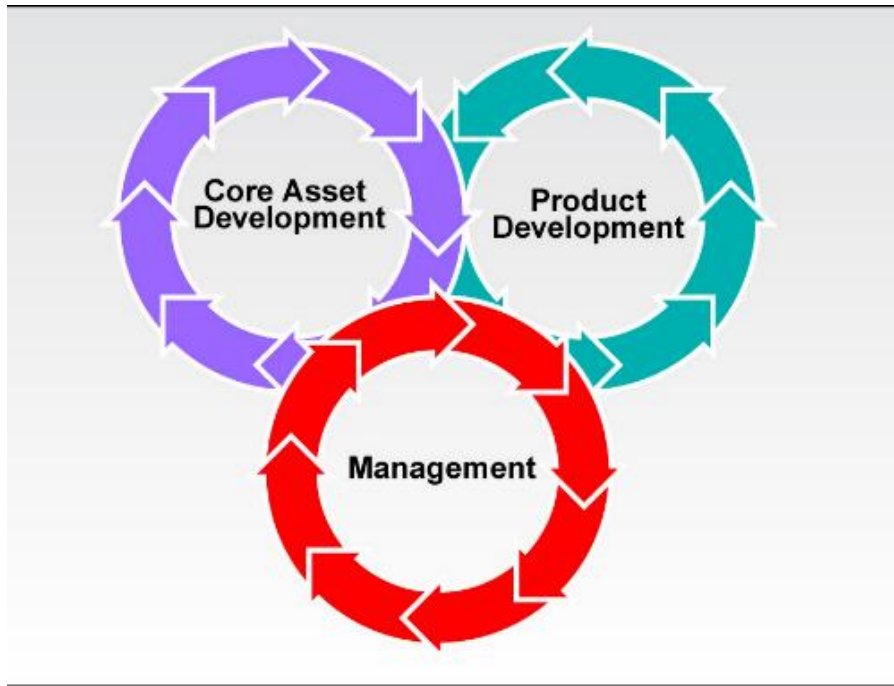
An accepted definition for software product lines is that a software product line is a set of software-intensive systems that share a common, managed feature set, satisfying a particular market segment's specific needs or mission and that are developed from a common set of core assets in a prescribed way (Clements and Northrop, 2002).

### 2.1.1 Software Product Line Essential Activities

Software product lines involves three essential activities, as shown in Figure 2.1: Core Asset Development (CAD), Product Development (PD), and Management (Clements and Northrop, 2002). These activities are detailed as follows:

**Core Asset Development.** The goal of this activity is to establish a production capability for the products (Clements and Northrop, 2002). It 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.

The activity of core asset development is iterative, and it is influenced by the situational context and existing constraints and resources. This context influences the way the core assets are produced. According to (Clements and Northrop, 2002) some contextual factors can be listed: *product constraints*, such as commonalities, variants, behaviors; *production constraints*, how and when the product will be bring to market, and other questions like this may drive decisions about variability mechanisms used.

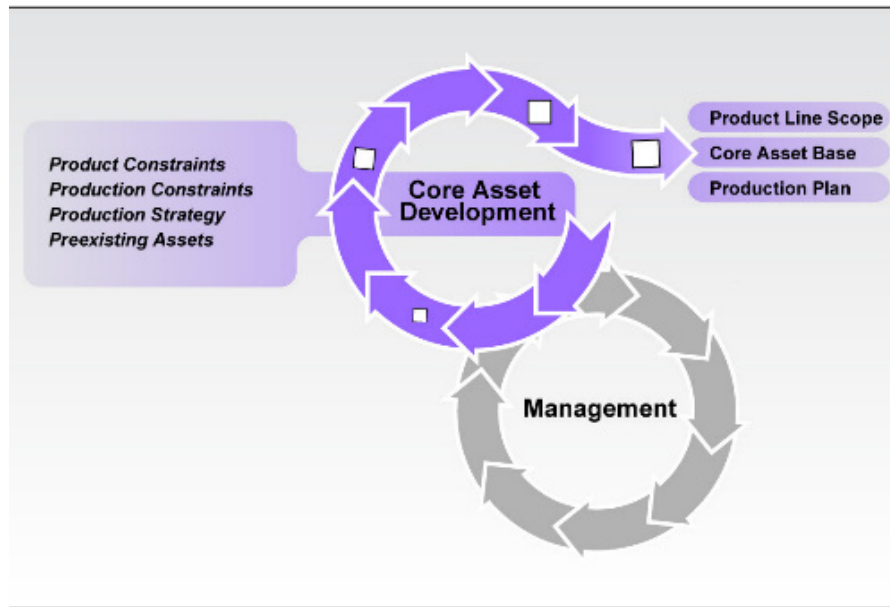


**Figure 2.1** Software Product Lines Essential Activities (Clements and Northrop, 2002)

One of the foundation concepts for the core asset development is the variability of it. Since core assets need to be compatible, and 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 it may be part. The matter of variability management will be further discussed in Section 2.1.2. Because of the variability inside each core asset, it is important to exist any sort of a guideline to guide the use of one asset inside 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. Figure 2.3 shows the production plan. This activity (Core Asset Development), is also known as *Domain Engineering* or *Framework Engineering* for other authors.

**Product Development.** The main goal of product development is to create individual (customized) products by reusing the core assets. The product development activities depends on the outputs provided by the core asset development activity (the core assets, and the production plan). The relationship of the SPL essential activities, with the focus on product development 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



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

an obligation to give feedback on any problems or deficiencies encountered with the core assets, in order to avoid the product line decay (minimizing maintainability and evolvability of assets and product) and keep the core asset base healthy and viable. This activity is also known as *Application Engineering*.

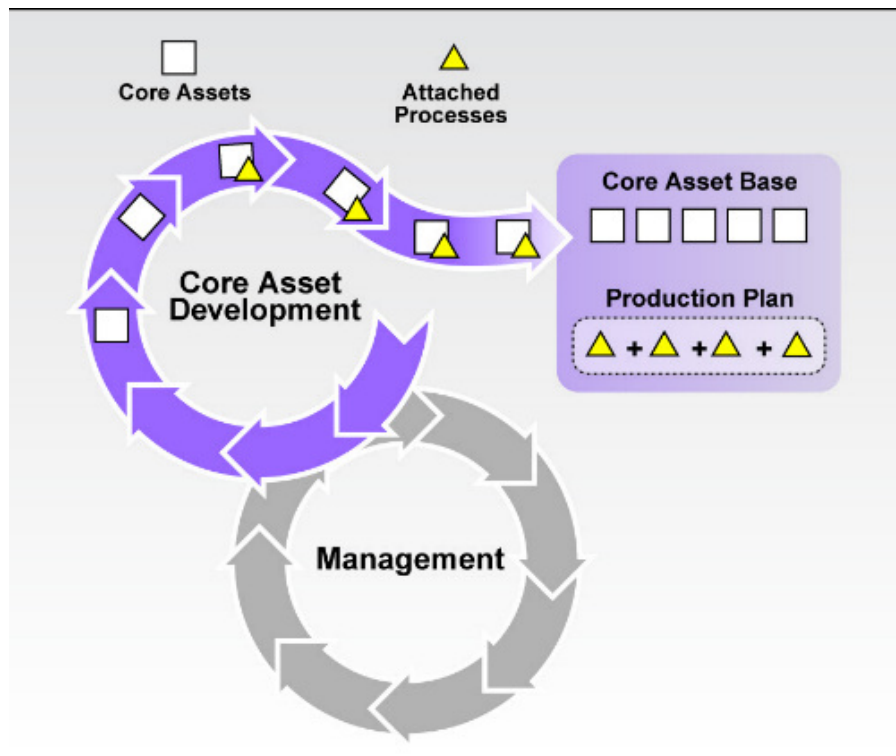
**Management.** Includes technical and organizational management, where technical management is responsible for the coordination between core asset and product development and the organizational management is responsible for the production constraints and ultimately determines the production strategy.

### 2.1.2 Software Product Line Variability Management

Variability is the ability or the tendency to change. 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 created and defined. To successfully introduce software product lines concepts in a software development environment, the notion of *variability* is extremely important.

A software component (or any other software piece) can have different behaviors (with different implementation/specification), and based on that, the variability is identified.

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



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

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 a product line (Pohl *et al.*, 2005a).

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

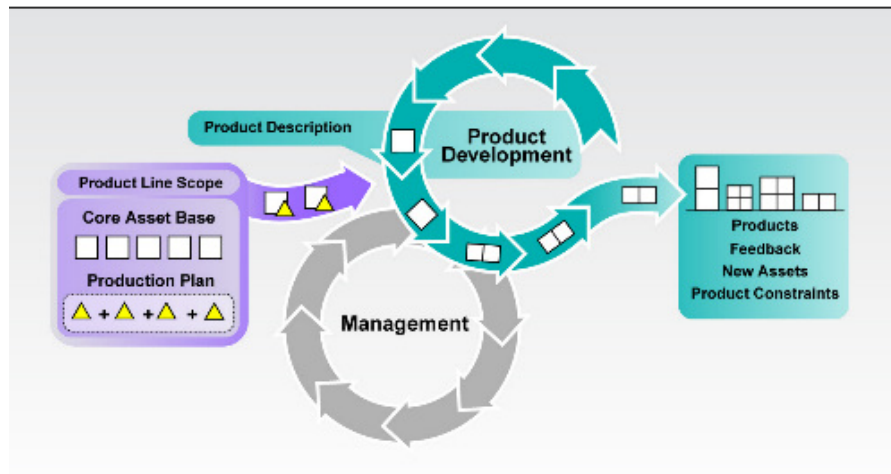


Figure 2.4 SPL Product Development (Clements and Northrop, 2002)

### 2.1.3 Software Product Line Adoption Strategies

In order to an organization to switch to product line engineering, there are some existing adoption strategies according to (Pohl *et al.*, 2005b). An organization considering to migrate to product line engineering generally has products already on the market and generally are under economic pressure. This pressure is driven by the need to produce the next products better and faster in order to maintain or gain the actual market share of the company. Software product lines is exactly the solution to both goals: faster development, better quality and less Time-To-Market.

The adoption of software product line engineering is started by defining which transition strategy will be used. In the following, 4 (four) major transition strategies are described (Pohl *et al.*, 2005a):

- **Incremental Introduction.** As the transition name says, the incremental introduction starts small and expands incrementally. This expansion may occur in two dimensions: (a) *Expanding organizational scope* from a single group, other groups are added incrementally after the first group success; (b) *Expanding investments* starts with a small investment and then incrementally the investment is increased according to the success.
- **Tactical Approach.** The tactical approach consists of introducing product line engineering concepts in sub-processes and methods partially, starting from the most problematic sub-processes.

- **Pilot Project Strategy.** The activities and process have to be set in order to start a pilot project, which can follow different ways, such as, starting as a potential first product, starting as a toy product, starting as a product prototyping (since engineering rules for prototypes are less strict than for standard products).
- **Big Bang Strategy.** In this strategy, software product line engineering is adopted for all new products at once, by developing the core assets and platform, and after deriving products from the core asset base.

## 2.2 Software Evolution

After the first NATO conference on software engineering (Naur and Randell, 1969), among the many software development activities, *maintenance* was considered a post-production activity, i.e., after the delivery and deployment of the software. This view, has been shared by some processes for software development, such as the waterfall process (Royce, 1970). According to the waterfall process, maintenance phase is the last phase of the software life-cycle, after the software deployment. In the maintenance phase, only bug fixes and minor changes were supposed to take place. This classical view on maintenance has governed the industrial practice in software development and is still in use today by several companies (Mens and Demeyer, 2008).

It took a while before the software engineers have realized the limitations of this model (waterfall-like models), given the fact that the separation in phases were too strict and inflexible. Generally, it is unrealistic to assume that all requirements are known before starting the design phase of the software (in many cases, the software requirements continues to change until the end of the software life-cycle).

Therefore, other attempts to a more evolutionary process model were proposed, such as the change mini-cycle (which introduced new important activities such as change impact analysis and change propagation) (Yau *et al.*, 1993), and also the studies of Manny Lehman, with his *laws of software evolution* (Lehman, 1980).

This was probably the first time that the term *software evolution* was used in contrast with post-deployment activity or software maintenance (Mens and Demeyer, 2008). But nowadays, software evolution has become a very active and well respected field in software engineering, and the terms *software evolution* and *software maintenance* are often used as synonyms. In this dissertation, the term *software evolution* will be used as opposed to software maintenance, because of the negative connotation of the latter term, as identified by Mens (Mens and Demeyer, 2008).



Inside evolution management, it is possible to identify some important research themes:

- **Change Management:** Change management is responsible for ensuring that the appropriate practices are in place to control the product line evolution. From a strategic perspective, change management is needed to guide the long term health of the organization's assets and products. An organization's ability to quickly respond to product opportunities depends in part on its ability to manage different evolving assets and products configurations (Atkinson *et al.*, 2002).

Change management is considered the heart of evolution, since every change has to be controlled and its impacts analyzed in a systematic way in order to guarantee the assets integrity and consistency (Atkinson *et al.*, 2002), (Atkinson *et al.*, 2000).

- **Build Management:** All activities related to build generation, which defines different types of builds (private system build, integration build and release build) and provides the steps to accomplish the build generation.
- **Release Management:** Release management is the process through which software is made available to, and obtained by, its users (van der Hoek and Wolf, 2003). It is an important part of the overall software process and mainly on product lines, because all releases of products inside a product line are composed of some core assets that can be or not developed and released independently from each other, and thus, needs to have specific activities and to support it.

### 2.2.1 Laws of Software Evolution

The laws of evolution were one of the first steps to the understanding of the software evolution area. In 1974, Lehman started to formulate the laws of software evolution, based on behaviors and observations. The laws are believed to apply mainly to monolithic, proprietary software, and the laws are predict that change is inevitable and not a consequence of bad programming.

The term *E-type software* is introduced and means a software system that solves a problem or implement a computer application in the *real world* (Lehman, 1980). The laws are summarized as follows.

**Continuing Change.** An *E-type* program that is used must be continually adapted else it becomes progressively less satisfactory.



**Increasing Complexity.** As an *E-type* program is evolved, its complexity increases unless work is done to maintain or reduce it.

**Self Regulation.** The *E-type* program evolution process is self regulating with close to normal distribution of measures of products and process attributes.

**Conservation of Organizational Stability (invariant work rate).** The average effective global activity rate on an evolving system is invariant over the product life time.

**Conservation of Familiarity.** During the active life of an evolving program, the content of successive releases is statistically invariant.

**Continuing Growth.** Functional content of a program must be continually increased to maintain user satisfaction over its lifetime.

**Declining Quality.** E-type programs will be perceived as of declining quality unless rigorously maintained and adapted to a changing operational environment.

**Feedback System.** E-type Programming Processes constitute Multi-loop, Multi-level Feedback systems and must be treated as such to be successfully modified or improved

Besides the laws, Lehman also defined two evolution dimensions, described in next Section.

### 2.2.2 Software Evolution Dimensions

There are two distinct dimensions in software evolution, according to Lehman ([Lehman et al., 2000](#)).

The first dimension is the **what and why** dimension, which focuses on software evolution as a scientific discipline. It studies the *nature* of the software evolution phenomenon, and seeks to understand its driving factors, impacts, and so on.

The second dimension is the **how** dimension, which focuses on software engineering as an engineering discipline. It studies the more pragmatic aspects of software evolution that aid the software engineer in his daily tasks. Hence, this dimension focuses on technologies, methods, processes and tools to manage software evolution.

Software evolution can also be categorized by the type of changes that are being performed into four different “dimensions” according to ([Lientz and Swanson, 1980](#)):

- *adaptive*, related to changes in the software environment;
- *perfective* which is the response to new user requirements;
- *corrective* where the focus is to fix errors;

- *preventive* where the objective is to prevent problems in the future.

The adaptive, perfective and corrective activities are *reactive* evolution management activities, since it concerns with managing the arrived changes and controlling them in order to maintain the stability and integrity of the systems. On the other hand, preventive activities can be seen as a *proactive* evolution management activity, since the efforts are on improving the evolvability, and preventing future problems.

The *how* dimension and the *reactive dimension* are the dimensions followed in this dissertation, because of the pragmatism of the *how dimension* (in terms of support for practioners on the daily tasks) and the reality of changing requirements and environments of the *reactive dimension*. Those points are strongly present in software product lines, and the relation between software product lines and evolution management is described in next Chapter.

## 2.3 Software Product Lines and Evolution Management

In terms of evolution, a product line is a continuously evolving organism, and for that, evolution should be managed properly to achieve all benefits of this approach. Evolution in a product line is complicated by the fact that an asset is shared among products, and any change in this asset may affect on several products (McGregor, 2003). This makes evolution management more challenging than in traditional single software development (Pussinen, 2002).

### 2.3.1 Forces for Change

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

**External forces** are one impetus for change 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 a the assets they provide to the product line.

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.** The Core Asset development exert evolutionary force on product development by providing new versions of assets and additional variants. The more frequent these releases are more product development resources will be consumed 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.
- **Product Development.** Product development exert evolutionary force 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 exert change forces by requesting a product specific asset to incorporate the core asset base, becoming a core asset reusable by other products.
- **Management.** Management exerts evolutionary forces on the core asset development by updating and adjusting the business plan for the product line. Core asset development responds to these forces by updating the existing core assets or creating new ones. Management also exerts evolutionary forces on the product development, by modifying the business case and the product line scope, and by doing that, the products release order may be re-evaluated.

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

### 2.3.2 Evolution Propagation

In traditional single-system development, the changes done to an artifact may have impacts on other artifacts of the system. In the software product line context, where change 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 2.3.1) 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 4 different directions, as (Atkinson *et al.*, 2002) defined.

- **Internal Propagation (Core Asset Development).** 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).** When a product change only has effects on specific products, 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 should be propagated back to the core asset development level (feed backing). Also, when a product specific asset should become a core asset (so other products can benefit from it), the product specific asset is propagated to the core asset development level, and integrates the core asset base.

Other differences between single-systems development evolution management and software product line evolution management, taking in consideration the disciplines of change, build and release management, are described in the next sub-section.

### 2.3.3 Evolution of Single-Systems vs. Evolution of Software Product Lines

In Table 2.1, we try to map the differences between evolution management disciplines of single-systems and the evolution management of software product lines.

Evolution Management Discipline	Single-System Engineering	Software Product Line Engineering
<b>Change Management</b>	Typically, in single-system development, changes performed to the system, need to be analyzed and performed, but its impacts are limited to the system being developed.	In software product line engineering, changes performed to any asset, may have a bigger impact, since it can affect every product that (re)uses the asset. Besides the change impact, the existence of two development levels (core asset development and product development) makes change management harder because the changes may need to be propagated from one level to another.
<b>Build Management</b>	The build generation of a system compiles the source code, creating the system executable file (generally in the binary form).	Before compiling the source code, the product build has to take in consideration the variability of the assets used. Before building a product, the variability of all used core assets must be resolved.
<b>Release Management</b>	Release management of single-systems generally involves the release planning and the release publication, with the release notes.	Besides the release planning and publication (core asset plus product plan or product), the existence of two development level, adds a new release level also. Core assets need to have their releases managed (to product development). From the product release, changes can be propagated back to core asset development (in case the changes were performed in the product development level).

**Table 2.1** Differences between single-system development and Software product lines

### 2.3.4 Challenges

Every difference from single-system engineering, cited in Table 2.1 can also be considered as challenges.

Thus, those challenges involve different solution spaces, such as technical issues, managerial and processes issues involving these themes. Nevertheless, this dissertation focus is on the process issues of evolution management of software product lines.

## 2.4 Chapter Summary

Software Product Lines (SPL) is an approach to software reuse that during the last years has proven its applicability in a broad range of situations, producing impressive results (Weiss *et al.*, 2006). To achieve all software product lines benefits, three essential activities must be followed: Core Asset Development, Product Development and Management.

Software evolution is the software engineering area that studies the phenomenon of software change, its impacts, drivers and contexts. As one of the first key developments in this area, Lehman defined the software evolution laws. The evolution laws can be summarized as follows: *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. The changes increase the entropy and complexity of the

program. Lehman also defined two evolution dimensions: the *what and why* dimension, which studies the evolution phenomenon and seeks to understand its driving factors, and impacts; and the *how* which studies the pragmatic aspects of software evolution as an engineering discipline, to aid software engineers in their daily tasks.

Next Chapter presents a systematic literature review performed in order to identify the existing processes and approaches to manage evolution in software product lines. All the steps followed in the systematic literature review are described in details in the next Chapter.

*“But let every man be quick in hearing, slow in words and slow to get angry.”*

James 1:19

# 3

## A Systematic Literature Review on Software Product Lines Evolution Management

Software Product Lines (SPL) offer the potential to reduce the time of building one or more software systems and increase the quality of the products maximizing reuse. However, compared to single systems development, the evolution management of assets and products in the SPL becomes more complex and challenging. In order to address part of these challenges, some approaches have been proposed by academy and industry.

This systematic review is an attempt to identify the available approaches, and verify how they deal with SPL evolution management and disciplines such as release management, change management, and build management. We believe that this systematic review contributes to understand the state-of-the-art in the area, providing insights to help choosing among the existing approaches, and pointing out directions for future research. Thus, the purpose is to review the SPL evolution management field to summarize empirical evidence about its state-of-the-art. This analysis was performed through a systematic review procedure, which aids in assuring the validity of the conclusions that are extracted from the individual studies (also known as primary studies) ([Kitchenham, 2007](#)).

The remainder of this Chapter is organized as follows. Section [3.1](#) introduces the concepts and benefits of systematic literature reviews. Section [3.2](#) presents the planning phase of the review, and the research questions it proposes to answer. Section [3.3](#) discusses the conduction procedures, along with the search strategy used and the approaches selected to be considered in this review. Section [3.4](#) presents the results of the analysis. Section [3.6](#) describes the related work and Section [3.7](#) summarizes this Chapter.

## 3.1 Systematic Literature Reviews

In this Chapter, we will address *systematic literature reviews* as *systematic reviews*. Among the advantages of systematic reviews, is that the well defined methodology makes it less likely that the results are biased, and provides a greater scientific value than conventional literature reviews (Kitchenham, 2007).

Some other features differentiate systematic reviews from conventional literature reviews, such as the definition of a review protocol that specifies the research question being addressed, the search strategy documentation in order to other readers assess their rigor, completeness and repeatability. Systematic reviews also requires inclusion and exclusion criteria as an evaluation form of each primary study. In this systematic review, we followed Kitchenham's guideline (Kitchenham, 2007).

The phases of a systematic review, according to (Kitchenham, 2007), are:

- **Planning**, where the need for the systematic review is explicated and the review protocol for the review conduction is defined;
- **Conduction**, where the review is conducted according to the protocol prior created, by selecting the primary studies, extracting and synthesizing data from the primary studies; and
- **Reporting**, in which the review results are reported and circulated to interested parties.

According to this definition, each one of the main phases is further described in the following sections.

## 3.2 Planning

In the planning phase of a systematic review, the review protocol is defined, and the most important item of the review protocol is the research question, since it drives the whole review process (Kitchenham, 2007).

### 3.2.1 Question Structure

We discuss this review from three different viewpoints: *population*, which are the people affected by the intervention; *intervention*, which is usually the software methodology/procedure that address a specific issue; and *outcomes*, the results of this review.



- **Population:** The population of this review is the SPL Evolution Management and the SPL area in general.
- **Intervention:** This review will search for indications that SPL evolution management process models and approaches fully address the existing challenges and issues.
- **Outcomes:** This review outcomes a summarized current state of evolution management in SPL describing the approaches main advantages and drawbacks.

### 3.2.2 Research Questions

It is important that the notion of software development process models, methods, techniques, and approaches are clear in order to understand the next steps of this systematic review.

A software process can be defined as a *coherent set of policies, organizational structures, technologies, procedures, and assets that are needed to conceive, develop, deploy, and maintain a software product* (Fuggetta, 2000). On the other hand, software development methods and techniques are *guidelines on how to use a certain technology and accomplish activities* (Fuggetta, 2000).

The term *approach*, used in this systematic review, can comprise these concepts of processes, methods, and can also be an informal approach, such as an experience report and best practices.

This systematic review tries to answer the following question: *How evolution is being managed in SPL?*

In order to improve the review clarity, this question is further divided into sub-questions (SQ) comprising the disciplines inside SPL evolution management. Each sub-question and its rationale is described next:

**SQ1.** Do the approaches deal with SPL configuration identification? How?

According to (SWEBOOK, 2007), configuration identification aims at identifying the items to be controlled, and establishing identification schemes for the items and their versions. The configuration identification is a key task of evolution management since the most effective way to manage the evolution of the whole product is to manage the evolution of each configuration item.

In the context of SPL, configuration identification could means (i) the identification of each part (configuration item) of each asset or product; and (ii) could also means the

identification of what is a core asset and what is not. The latter can be addressed by earlier stages in the SPL development such as scoping and domain analysis, where it is defined, for example, whether an asset will be developed as a core asset or a product specific asset.

Since the second meaning (ii) of configuration identification is more related to SPL scoping techniques than evolution management itself, it was left out of the sub-question scope. Thus, this question covers the configuration identification in terms of what will be formally controlled inside each core asset or product.

**SQ2.** Do the approaches enable *multi-level* instantiation of assets? How?

This question is related to the variability decisions in the core assets. Since most core assets have inside its structure some open variability decisions, it is important for the SPL evolution management approach to support the core asset instantiation (variability decision choices) in more than one level. As an example, there is a SPL where there are two different **core asset** development levels, in the first level we have all variability decisions still open, and in the second level, part of the variability is decided (configured), and only then the asset is used in the product development level.

This instantiation, which can occur in different levels, is popular specially in product populations environments (van Ommering, 2000). This partial instance is the first level of instantiation of the core asset. From the partial instance, one may continue deriving more instances, what gives further levels of configuration. Some kind of support is necessary to control these multi-level instantiation of core assets like what activities should be performed, or what tools should be used.

**SQ3.** Do the approaches deal with SPL release management? How?

Release management is the process through which software is made available to, and obtained by, its users (van der Hoek *et al.*, 1997). It is an important part of the overall software development process. In SPL, two levels of releases can be identified: *core asset releases* (targeted to the product development team) and *product releases* (generally targeted to market or end users). Each core asset can be developed and released independently from each other, since they are generally (re)used in more than one product, and thus, need to have specific activities to support the release procedures. The composition of the core assets releases may compose products, that need to be released as well following specific procedures.

**SQ4.** Do the approaches deal with SPL change management?

---

Changes in SPL, when left uncontrolled, become very expensive and complex to manage, because of problems such as shared assets among products and conflicting change interests driven by each product.

This sub-question is further subdivided into the following sub-questions, in order to address different topics inside the change management discipline.

**SQ4.1** Do the approaches deal with variability evolution (adding, removing and changing)? How?

An important issue inside change management is the variability evolution. Adding, removing or changing variation points, changing dependencies among assets, changing variability binding times are examples of activities that should be driven by specific change management practices or guidelines, in order to guarantee the consistency and integrity of the whole SPL.

Most of the times, these procedures follow the same rules and steps as regular changes inside an asset, but they require special attention.

**SQ4.2** Do the approaches deal with assets change propagation between core asset and product development? How?

One major question, when there are change requests to product specific and core assets, is whether the correction is going to be made at the level of the core asset development, or at the level of an individual product (Atkinson *et al.*, 2002). Depending on these decisions, the changes performed will probably need to be propagated in a given moment.

Propagation of changes between core assets and product development is important because the changes can be replicated between these two layers (core asset and product development) minimizing some of the change management issues described in section 3.4.4.

This is a very important point inside change management, since it is the *bridge* between core asset and product development.

Some guidelines, well defined activities, or any other sort of control should be provided to properly manage this propagation of changes, and answer questions the ones listed next:

- How do changes made in the core asset are propagated to products that (re)uses these assets?

- How do changes made in a core asset instance (product level) are propagated back into the core asset?

**SQ5.** Do the approaches deal with build management? How?

The build management procedures and tools are responsible to guarantee the reproducibility of the generated build, and goes beyond that, by defining build strategies, defining exactly what part of the system is targeted to build (Bay, 1999).

Generally it exists some kind of configuration or parameterization (conditional compilation directives, configuration files, aspect orientation, simply if-else blocks, etc.), to address the multiple variants of certain assets (Gacek and Anastasopoulos, 2001).

One could relate build management with quality assurance, which is not the focus of this research. Furthermore, the build management discussed in this research includes not only the selection of the variants to be built, but includes the strategies and procedures of the build itself (such as tools, steps, etc.).

### 3.2.3 Threats to Validity

The main threats to validity identified in this review are described next:

- *Selection bias*: The studies selection from web search engines and key digital libraries do not ensure that all SPL evolution management approaches were reviewed. Possibly, relevant approaches were excluded from this review. Thus, for reducing this threat, we searched for referenced papers.
- *Publication Bias*: Most of the organizations running SPL, probably have some kind of approach to manage evolution. Some of those existing approaches are not published in the academia, or are considered as confidential to the organization business. To minimize this threat, we decided to include the last inclusion criteria “Strategies and experiences reports” in Section 3.2.4.
- *Research Questions*: It is likely that some questions defined in the protocol may not be so relevant. They are not the unique questions to address in this area. To minimize this threat, several meetings were held in order to discuss the questions.
- *Data Analysis Bias*: Sometimes the researchers experience may influence the results, and sometimes a reduced set of papers or a unique paper addressing specific issues, may also influence. Aiming to avoid it, we had a set of discussion meetings with the project members and experts in the area.

### 3.2.4 Inclusion and Exclusion Criteria

According to (Kitchenham, 2007), it is necessary to define inclusion and exclusion criteria for the studies selection, in order to reduce the likelihood of bias. These criteria should select the primary studies.

The inclusion criteria defined for this review were:

1. *Approaches that offer any kind of support to SPL evolution management:* SPL evolution management can comprise different disciplines (McGregor, 2007). Approaches which include activities regarding any of the disciplines previously cited will be considered as a primary study in this review. The approaches will be considered primary studies even if it covers only core assets development, which is the phase where the core assets are defined, designed and built.
2. *Strategies and experiences reports:* All studies which contains experience reports, or defines strategies to deal with any topic of SPL evolution management will be considered as a primary study.

If the inclusion criteria is present, the approach will be considered in the review. As exclusion criteria, we defined the following:

1. *Approaches that deal exclusively with Software Configuration Management (SCM) or Component-based development (CBD) specific issues:* Some studies were conducted in this direction, from the identification of component upgrades and replacement problems (Brereton, 1999; Crnkovic and Larsson, 1999; Voas, 1998) to proposed frameworks to make component upgrades less painful (Cook and Dage, 1999). Since these questions are not the main target of this review, any work in this direction will not be considered.
2. *Lack of detailed information about SPL evolution management:* Any study that does not have detailed information about evolution management will not be considered in this review. Even if a study discusses about evolution, but it does not go into further details, it will not be considered.

## 3.3 Conduction

The search strategy is the plan of finding as many primary studies relating to the research question, using an unbiased search strategy (Kitchenham, 2007).

---

For a better documentation of the systematic review, some information such as the data extraction forms of each primary study are available at the systematic review website<sup>1</sup>.

### 3.3.1 Search Strategy

This strategy is divided in defining the data sources and search query strings, searching for the primary studies and selecting the studies according to the inclusion and exclusion criteria previously defined.

### 3.3.2 Query Strings

A set of keywords was raised, however after a brief preliminary search for primary studies, it was realized the need to have the set of keywords calibrated in order to refine the results, due to the huge amount of irrelevant results gained. The keywords similar nouns and syntactic variations (e.g. plural form) were also used.

Here we present the final set of keywords used. They are:

- *Evolution*
- *Evolution Management*
- *Software Configuration Management*
- *Change Management*
- *Release Management*
- *Build Management*
- *Variability Management*
- *Configuration Identification*

All terms were combined with the term "Product Line" and "Product Family" by using Boolean AND operator.

The strings had been generated, as shown below:

1 evolution AND "evolution management" AND ("software product line" OR "product family" OR "SPL")

---

<sup>1</sup>Systematic review web site - <http://www.cin.ufpe.br/~thbo/systematicreview/index.php>

---

- 2 "software configuration management" AND ("software product line" OR "product family" OR "SPL")
- 3 "change management" AND ("software product line" OR "product family" OR "SPL")
- 4 "release management" AND ("software product line" OR "product family" OR "SPL")
- 5 "build management" AND ("software product line" OR "product family" OR "SPL")
- 6 "variability management" AND ("software product line" OR "product family" OR "SPL")
- 7 "configuration identification" AND ("software product line" OR "product family" OR "SPL")

### 3.3.3 Data Sources

The research focus was initially on important journals, since the most important work are usually published on journals. The search was done through keyword matching at the journal website and/or access portals.

After that, studies were searched on important conferences proceedings. The search methodology was the same as the one used for the journals, searching on conferences portals and/or websites with the defined set of keywords.

Next, other relevant electronic sources was also searched to guarantee a more exhaustive search (Brereton *et al.*, 2007), and for that, theses and technical reports were also included. Both the keywords used and the list of the searched journals, conferences and other electronic sources are listed in Appendix C.

### 3.3.4 Studies Selection

The approaches selection process was performed by four M.Sc. candidates and two Ph.D. At the end of the data source collection, 20 potential primary studies were identified as possible choices for further analysis based on the inclusion criteria. Among the studies were 16 papers, 2 Ph.D. theses, 2 technical reports, and 1 book. The studies were analyzed after a full text reading or just the title and abstract, in case of papers which the content was clearly not related to the research question. After the exclusion criteria were applied,

---

7 approaches were selected (detailed next). The selected approaches analysis was based on 9 papers, 1 book, and 1 technical report which are considered primary studies in this review (Budgen and Brereton, 2006). Some SPL approaches were not considered in this review because of the exclusion criteria defined in the previous Section. Some excluded approaches are described next:

- **PULSE:** The Product Line Software Engineering (PuLSE) was developed at Fraunhofer Institute for Experimental Software Engineering (IESE) with the purpose of enabling the conception and deployment of software product lines within a large variety of enterprise contexts (Bayer *et al.*, 1999).
- **PECOS:** PECOS is a collaborative project between industrial and research partners that seek to enable component-based development of embedded systems, specifically "field devices" (Genssler *et al.*, 2002).
- **RiDE:** RiDE is a domain engineering process focused on core assets development, and includes the steps of domain analysis, domain design and domain implementation (Almeida, 2007).
- **FAST:** The Family-Oriented Abstraction, Specification and Translation was developed in Lucent Technologies and its goal is to provide a systematic approach to analyzing potential families of products and develop facilities for the efficient production of any family member (Weiss and Lai, 1999).
- **PLUSEE:** PLUSEE is an approach to product lines that uses UML. It defines a domain model that depicts life cycle phases, views within each phase and meta-classes within each view (Gomaa and Shin, 2002).
- **KOALA:** Koala is a component-oriented approach for consumer electronics software. This approach was proposed to handle diversity and complexity of embedded software at an increasing production speed (van Ommering *et al.*, 2000).

The approaches selected to be considered in this review are described next in alphabetical order.

- **KOBRA:** *Komponentenbasierte Anwendungsentwicklung* (KobrA) is the German translation for component-based application development (Atkinson *et al.*, 2000) and was developed at the Fraunhofer Institute for Experimental Software Engineering (IESE) and it is a "ready-to-use" customization of PuLSE (Bayer *et al.*, 1999). Its analysis was based on (Atkinson *et al.*, 2002) and (Atkinson *et al.*, 2000).
-



- **MOHAN AND RAMESH:** This approach is a case study for identifying SPL change management best practices and possible patterns to deal with common changes scenarios. The analysis was based on (Mohan and Ramesh, 2006) and (Mohan *et al.*, 2008).
- **NICTA:** This study is an experience report from an Australian company, reporting some problems with change control in a SPL environment. NICTA stands for National ICT Australia. The evolution problems faced by NICTA, and the possible solutions to those problems are presented in this study. The analysis was based on (Staples, 2004).
- **PHONAK:** Phonak is a Swiss company that runs a SPL of PC-based applications, used by audiologists to fit hearing instruments to hearing-impaired customers (Kurmann, 2006). Phonak's focus is to obtain the benefits of SPL and agile development techniques to rapidly develop high quality software products. This study reports some Phonak's experiences with SCM and release management. The analysis was based on (Kurmann, 2006).
- **SEI:** SEI technical report discusses evolution of SPL assets. The report was developed at the Software Engineering Institute (SEI) and the focus is on change management and configuration management. The analysis was based on (McGregor, 2003) and (McGregor, 2007).
- **SRM:** SRM is a process, and a support tool, to effectively manage evolution through release management. The work is focused on component-based software release management, but it can be easily applied to SPL context. The analysis was based on (van der Hoek *et al.*, 1997) and (van der Hoek and Wolf, 2003).
- **TABORDA:** Taborda's study addresses planning and management of releases in the context of component-based SPL. The analysis was based on (Taborda, 2003).

### 3.3.5 Data Extraction

In the data extraction step, the main author read every primary study, and documented the answers to the research questions in the data extraction form. These forms can be found at systematic review website.

### 3.3.6 Data Analysis and Synthesis

In this systematic review, we synthesized data in a qualitative form, by a descriptive synthesis, as it is more natural in the software engineering field (Brereton *et al.*, 2007). The analysis and synthesis are reported, and results discussed in the next Section.

## 3.4 Reporting

In this Section, the results of the review are presented. Since the research question was further divided into a set of sub-questions, each sub-question is discussed, analyzed and conclusions are drawn in the next Sections.

### 3.4.1 Configuration Identification

No approaches addressing configuration identification in the context of SPL were identified and considered in this review. However, in practice, there are even models, such as the CMMI, which requires activities like that in order to explicitly know what are the items in the product that will have its changes be formally controlled (Chrissis *et al.*, 2003). Thus, this activity could be considered as the foundation of the change management process, since to manage the changes of one system, it is needed to know all the items of the system that will be managed.

From single-system engineering to SPL, some differences arise and must be addressed to better fit SPL. Some sort of guideline should exist in order to facilitate the identification of configuration items, specially considering the differences between core asset and product development.

This is still an open issue for future development of the SPL evolution management field.

### 3.4.2 Multi-Level Instantiation

No primary studies regarding this topic were found, showing an opportunity to the academia to perform more studies towards this direction. This question is of extreme importance for approaches such as Product Population (van Ommering, 2000) or hierarchical product lines.

When there are many levels of assets development, generally there is also some sort of branching support, or other instantiation support to guarantee that the assets are well

organized, and changes can flow through the instances (enabling change propagation). Some well defined activities may also support this area, or even a support tool.

Probably SPL companies have some kind of support to this area, but no published study was found with detailed information.

### 3.4.3 Release Management

Analyzing all the approaches, we could identify that four of them have some kind of support for release management.

The first approach is the one where Van der Hoek and Wolf ([van der Hoek and Wolf, 2003](#)) defined a release management process and also a tool called Software Release Manager (SRM) to support this process, focused on CBD. The SRM is based on two key notions:

- The actual location of each component is transparent.
- Dependencies among components are explicitly recorded (dependency tracking).

A set of requirements to this process and tool were raised addressing both component producers (core asset developers) and potential consumers (product developers). Among these requirements, the most important requirements for the component producers were: (i) Dependencies should be explicit and easily recorded; (ii) The scope of a release should be controlled. As for the component consumers, the main requirement were: (i) Descriptive information should be available; (ii) Location transparency; (iii) A component and its dependencies should be retrievable, preferable as one single file.

The next approach presented is strictly from the SPL field, and it is an experience report from PHONAK ([Kurmman, 2006](#)), describing three release management strategies that were in use in the company and were successful:

1. *Align platform and product releases*: The effort of releasing the platform independently from a product is not efficient because verification and validation is often difficult, and it adds an unnecessary Time-To-Market (TTM) delay on new features, since they need to be released as a platform release, in order to be reused in a product release.
2. *First release of new feature with one product*: First release of any new feature is always with only one product because, during development time, requirements may change and they need to be implemented quickly. Adapting it to other products may throw them off schedule or invalidate the feature variation validation efforts.

3. *Releasing source code rather than binaries*: Kurmann ([Kurmann, 2006](#)) identified that there was no purpose or need in releasing the binaries of their core assets, since their build process was fully automated and the binaries could be generated at any time.

These strategies are important because they were extracted from industry practical experience. Some of the strategies, such as the alignment of the platform and the products release may cause a serialization in the release process, because other products may have to wait the first release of the platform to be released. There is a need of a good release management team to coordinate these schedules and guarantee that no product will be impacted because of problems like this.

Following the same line of work, NICTA experience report ([Staples, 2004](#)) outlines some challenges related to evolution management and the importance of planning, coordinating, tracking, and managing the impact of change to software assets. One problem was related to release management and is the different product release constraints. Products in a SPL often have different release constraints, such as schedule and quality constraints. Since these products share core assets, a product release sometimes depends on conflicting release constraints of these shared assets. The solution for this problem was the SPL roadmap, which is a plan for changes to core assets. That can be useful to manage the expectations and plans of the products stakeholders. This roadmap can also plan for architectural changes.

Taborda's work goes further than the others in terms of release management. He states that for each new component release, there is a need to prioritize or negotiate the competing, potentially conflicting, requirements across the user-base and package the selected ones into the component's release plan ([Taborda, 2003](#)).

In order to manage this release plan, Taborda defines the Release Matrix. The introduction of the Release Matrix offers a useful technique to develop a master plan for the SPL that can help to consolidate and to align the individual release schedules of the different products and components ([Taborda, 2003](#)). The Release Matrix offers a means to coordinate and record each incremental step in the SPL evolution, and is presented as a generalization of release planning in complex reuse contexts. Each column in the Release Matrix represents a component release plan defined as the sum of the requirements it is to implement in that release. A row represents a product release plan that must be compatible with the component releases that the product is reliant upon. As a whole, the Release Matrix represents a master release plan that consolidates and synchronizes the individual release plans of the modified products and components ([Taborda, 2003](#)).

The Release Matrix could be used to provide an overview of the requirements' progress as they are implemented. The status of each component development can be presented in a similarly formatted reporting matrix that indicates when a product allocated requirements have been met and the integration and test of the product can proceed.

Even though the release matrix helps solving the problem of release planning, it is likely that this matrix starts to become difficult to use and maintain in a SPL scenario where there are hundreds of components, and each product comprises most of those components. Some kind of release matrix clusters may be interesting to organize the growth of the matrix.

### 3.4.4 Change Management

From the approaches selected and included in this review, four of them covered the subject of change management somehow.

Mohan and Ramesh detected three patterns of changes which were constantly repeated inside SPL, and for each pattern, a change management practice was recommended (Mohan and Ramesh, 2006), according to Table 3.1.

Issue Raised by Pattern	Recommended Practice
Interdependencies among changes in variants	Modularize changes and variation points
Increasing the degree of evolving variance	Modularize changes and variation points
Reinvented variations	Facilitate reuse based on knowledge sharing

**Table 3.1** Common issues raised x Recommended change management practice

Following the same line of Mohan and Ramesh's study, NICTA experience report (Staples, 2004) outlines some challenges related to SPL evolution management and the importance of planning, coordinating, tracking, and managing the impact of change to software assets. For each change control problem, a possible solution was proposed, based on practical experience. The problems identified and the solutions to them are described next:

1. *Changes to PLA Core Asset Variation Point:* A Product Line Architecture (PLA) change is a change to the core assets' interface, and so can force changes in all products that use the new version of these core assets. The solution proposed to this problem is the creation of a core CCB, which would be a primary mechanism to deal with change control. In the case of SPL, a core assets CCB is needed. This

core CCB would be responsible for the negotiation among product stakeholders, and help to solve the problem of different product release constraints.

2. *SPL Decay*: SPL decay occurs when new similar or identical functionality is implemented in the asset instance of a number of specific products. The proposed solution to address the decay problem, is continual vigilance to calculate the cost of a "core" functionality that is implemented in instantiated assets.

In a more complete study, SEI technical report on SPL assets evolution, change management is faced as the process responsible for ensuring that the appropriate practices are in place to control the evolution of the SPL (McGregor, 2003).

SEI's report also states that change management must address the change impact analysis, because it provides means of predicting which assets will be affected by a certain change. This analysis can involve: conducting a traceability analysis to identify impacted artifacts; identifying the interactions affected by the change; evaluating the effect of changes on assumptions and identifying new constraints and identifying regression tests.

An important notion that McGregor depicts is the notion of change dimension (McGregor, 2003). When variation (which is a possible kind of change) occurs, it changes the whole configuration of the SPL. Since the SPL configuration is changing, the configuration of specific products changes along with it.

Change management processes are being placed as one of the fundamental key disciplines in SPL by some approaches. One example for that is the the Kobra approach (Atkinson *et al.*, 2000, 2002) described next.

Kobra approach (Atkinson *et al.*, 2002), is basically a CBD approach, and according to CBD each component is an independent, reusable entity, with clear dependencies and provisions. Kobra's change management is based on this principle. The whole process of maintenance is based on changes, even though it is separated on two distinct processes: change management, which is responsible for the changes identification and rationales, and configuration management, which is responsible for how the changes are achieved (Atkinson *et al.*, 2002).

Change management is considered the way to drive evolution of the assets, and consequently the evolution of the SPL. Changes are treated as *first-class citizens* as well as their relationships to other assets, that means, every change or dependency relationship will be analyzed and carefully implemented.

### Variability Evolution

Inside SPL, variability is a key notion, and the way to deal with variability should be very well documented and formalized. However, no approaches were found in the direction of how to deal with changes in the variability, what steps should be taken, and what should be analyzed in order to guarantee that the change will not have a negative effect on the existing products, and assets.

### Change Propagation

When a core asset is instantiated in a certain product, and after that, some changes are made, it should exist some kind of support to propagate these changes back to the original core asset. By doing that, the next product to reuse this core asset, will reuse a version with those changes.

SEI's report considers that the SPL evolution management process has to comprise the promotion of assets to ensure that the specific assets that becomes a core asset still supports the original use that brought it into existence. It defines one guideline for promotions, which states that promoted assets should stay backwards compatible with their previous use (McGregor, 2007).

As one of the most complete approaches regarding this sub-question, Kobra approach describes four scenarios to manage change propagation.

- *Core Asset change integration*: Propagation of changes to other assets inside the core asset base to ensure that the core asset base is in a consistent state.
- *Product change integration*: Propagation of changes to other assets inside the product specific asset, to ensure that the product is in a consistent state.
- *Feed-forward change integration*: Propagation of changes from assets of the core assets base to other assets inside the product specific artifacts to update asset to a more recent and correct state.
- *Feed-back change integration*: Propagation of changes from assets of the product specific to other assets inside the core assets base to update assets to a more recent and correct state.

Each one of these scenarios shares a set of sub-activities like: change identification, change impact analysis and the change propagation itself.

### 3.4.5 Build Management

The Build management area is tightly related with release management, and could be considered as a required activity to release management procedures. SPL build strategies and build management experience reports could provide valuable information to SPL practitioners and researchers, but were not found in this review, showing that this area has still much to be studied and explored.

The decision model in KobrA could be related to build management, when a decision model is resolved, a resolution model is obtained, and could serve as an input to build management procedures. Since it only addresses the question partially, we did not consider as a primary study to build management.

Still in the build management area, there are some product derivation (instantiation) tools that deal with the automatic build generation of products. Tools such as Gears (Krueger, 2008), pure::variants Beuche (2008) offers great support for build automation and generation. These tools will not be included in this review since the focus of the review is on process issues.

Apart from that, no approaches were found detailing build management procedures, strategies or experiences.

### 3.4.6 Questions Summary

The disciplines covered by the research question and sub-questions are very important topics to the SPL evolution management field. Therefore, we could notice that no approach considered in this review comprehended all these areas, as we can see in Table 3.2.

Sub-question \ Approaches	KobrA	Mohan and Ramesh	NICTA	PHONAK	SEI	SRM	Taborda
SQ1 - Configuration Identification							
SQ2 - Multi-Level Instantiation							
SQ3 - Release Management			X	X		X	X
SQ4 - Change Mangement	X	X	X		X		
SQ4.1 - Variability Evolution							
SQ4.2 - Change Propagation	X				X		
SQ5 - Build Management							

**Table 3.2** Summary table of all sub-questions and approaches match

We may draw some findings from Table 3.2, from which we identify some disciplines which are covered by most of the approaches (marked with an X), and thus, we can conclude that those disciplines are more mature than others, such as change management



and release management. On the other hand, disciplines such as multi-level instantiation, build management, and variability evolution do not have much information sources and available studies, showing that these disciplines have still much space to grow.

Furthermore, among the primary studies included in this review, none of them had any kind of formal process definition for evolution management, indicating that the efforts on evolution management in SPL is still not mature, and organizations do not follow a defined process model for it.

It is very difficult to suggest any of the approaches to be followed or used, since we do not have a single approach comprising all (or most of the) disciplines of evolution management. For this reason, maybe a combination of the strong points of each approach is the most interesting “*composed approach*” to follow. From the research community perspective, we could notice that there is still much to be explored.

## 3.5 Lessons Learned

The lessons learned with this systematic review are described next:

**Number of disciplines being covered by the sub-questions.** This review covered various evolution management disciplines by the sub-questions. However, to improve the analysis and outcomes of the systematic review, one could reduce the number of disciplines and focus on only one or two. On the other hand, SPL evolution management approaches (as we could conclude in this review) are in a very reduced number comparing to other areas of SPL. In this sense, it is reasonable to include all these disciplines. In areas where there are lots of work available the ideal is to reduce the scope of the questions.

**Studies quality assessment.** In this systematic review we did not assessed the quality of the primary studies selected. In addition to general inclusion/exclusion criteria, it is considered critical to assess the “quality” of primary studies to provide more detailed inclusion/exclusion criteria and to serve as means to weighting the importance of individual studies when results are synthesized. The initial difficulty is that there is no agreed definition of a study “quality”, but it is suggested that quality relates to the extent to which the study minimizes bias and maximizes internal and external validity. Quality assessment instruments can be used in two different ways: (i) to assist primary studies selection (providing detailed inclusion/exclusion criteria); (ii) to assist data analysis and synthesis to identify whether quality differences are associated to different primary studies outcomes. In either cases, a specific form should be used to each kind of study, to document the quality assessment.

## 3.6 Related Work

Only a technical report from Pussinen, was found with the same purpose of evaluating the existing SPL evolution management approaches. Pussinen's survey, however, was not performed through a systematic review; it was a survey of some available points regarding evolution (Pussinen, 2002). Pussinen's survey had a different focus, it starts discussing about organizational models and assets evolution responsibility, and then discusses about the recording of assumptions and design decisions, which are both, important to guide the evolution and maintenance. It covers also evolution metrics and visualization of evolution traces. None of the topics covered by Pussinen's work was covered by this review, since the focus of this review was mainly on evolution management processes.

## 3.7 Chapter Summary

In this Chapter, it was presented a systematic review in which the main goal was to identify, among the relevant studies and work, the existing solutions of evolution management in SPL. We believe this study contributed to the field, by reporting the obtained results, and stating that the field has still much to be explored.

No approach included in this review had any kind of a formal process definition, furthermore, no approaches comprehended all evolution management disciplines listed. For this reason, the combination of the positive aspects of each primary study would be the best option for practitioners.

This study is the first systematic review performed in the SPL evolution management process area, and can serve as basis for the definition of a more complete and detailed process. Each approach analyzed had interesting insights about at least one, or more evolution management disciplines, and each of these approaches can be used for the creation of a more complete and formal process definition.

*“The only constant is change.”*

Heraclitus (535 BC / 475 BC)

# 4

## RiPLE-EM

The process proposed by this dissertation, RiPLE-EM, is part of a more general process called **RiSE Product Line Engineering (RiPLE)**, which concerns with the full software life-cycle for software product lines. In this context, RiPLE-EM concerns with the evolution management of software product lines, or, more specifically, the evolution management of the assets and products inside a product line.

This chapter introduces the concepts of RiPLE-EM by presenting the RiPLE-EM overview in Section 4.1 and describing RiPLE-EM disciplines in Section 4.2. RiPLE-EM usage scenarios are presented in Section 4.3, RiPLE-EM communication between core asset development and product development is discussed in Section 4.4. RiPLE-EM relationships with existing support tools are described in Section 4.5 and RiPLE-EM roles are presented in Section 4.6. RiPLE-EM activities summary is presented in Section 4.7 and the RiPLE-EM work products are listed in Section 4.8. The concepts of the Eclipse Process Framework (EPF) is discussed in Section 4.9 and the chapter summary is presented in Section 4.10.

### 4.1 RiPLE-EM Overview

A characteristic of RiPLE-EM is that is a process focused both on core asset and product development, in a release-oriented way. By release-oriented, we mean that for each core asset or product release, a new RiPLE-EM flow, focused on the evolution management activities is started.

RiPLE-EM has two different flows, one for each of the two essential activities of SPL engineering: Core Asset Development (CAD) and Product Development (PD). Figure 4.1 shows the main flow of RiPLE-EM.

Figure 4.2 illustrates RiPLE-EM CAD and RiPLE-EM PD in a macro view.

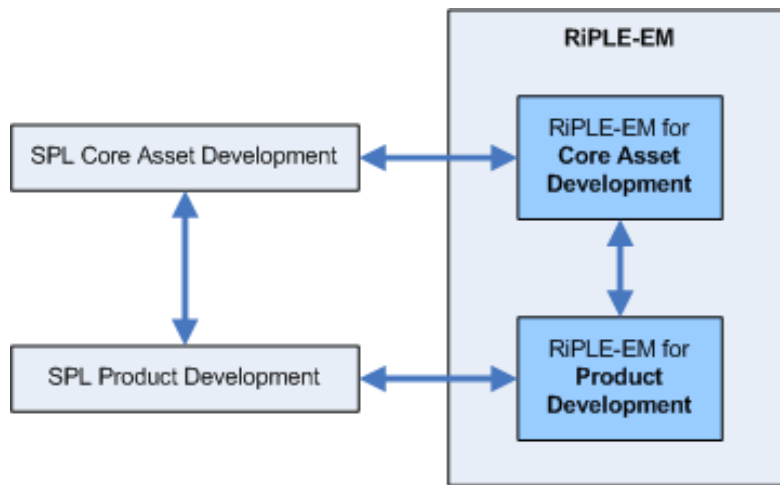


Figure 4.1 RiPLE-EM Main flow

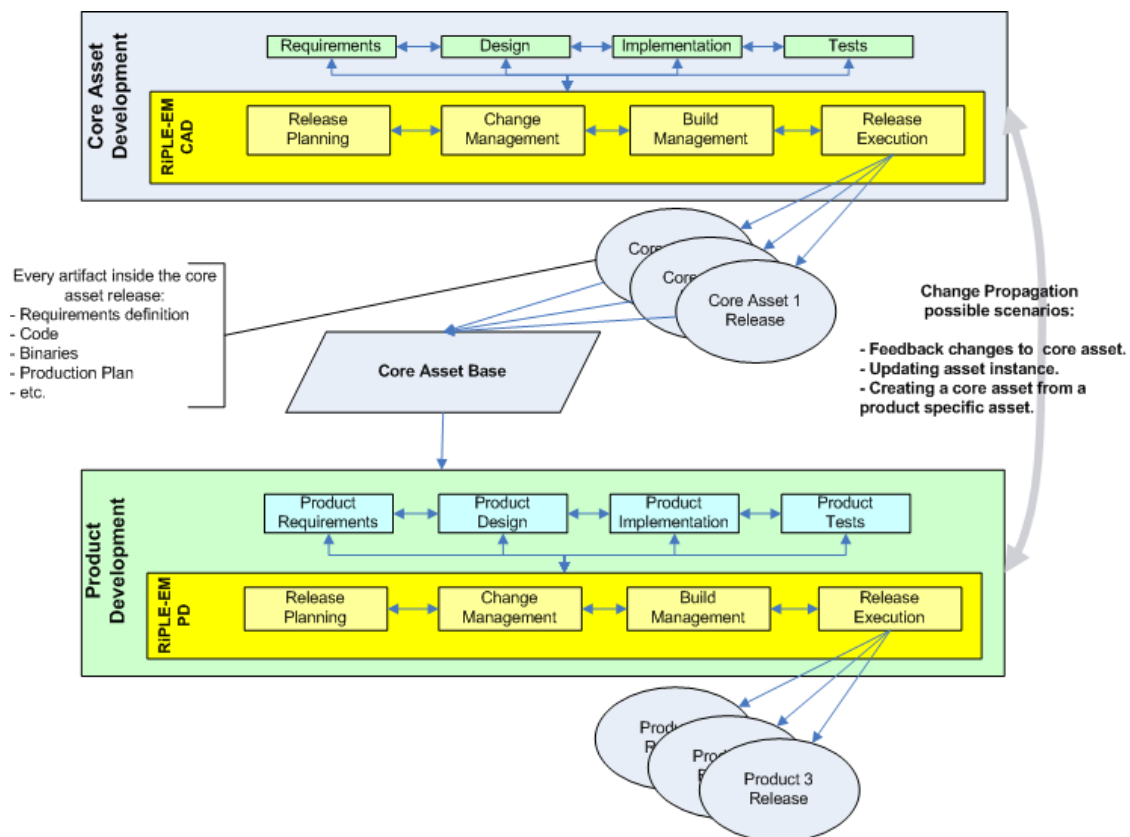


Figure 4.2 RiPLE-EM Big Picture

## 4.2 RiPLE-EM Disciplines

Both RiPLE-EM CAD and RiPLE-EM PD, when expanded, have activities regarding the following disciplines (described in Chapter 2).

**Change Management.** Changes in SPL, when left uncontrolled, become very expensive and complex to manage, because of problems such as shared assets among products and conflicting change interests driven by each product or feature.

Change management is the area responsible for ensuring that the appropriate practices are in place to control the products and assets evolution (changes), and is considered the heart of evolution management, since every change has to be controlled and its impact analyzed in a systematic way, in order to guarantee the assets integrity and consistency (Atkinson *et al.*, 2002). From a strategic perspective, change management is needed to guide the long term health of the organization assets and products. The ability to quickly respond to product opportunities, depends in part, on the ability of the organization to manage different evolving assets and products (McGregor, 2007).

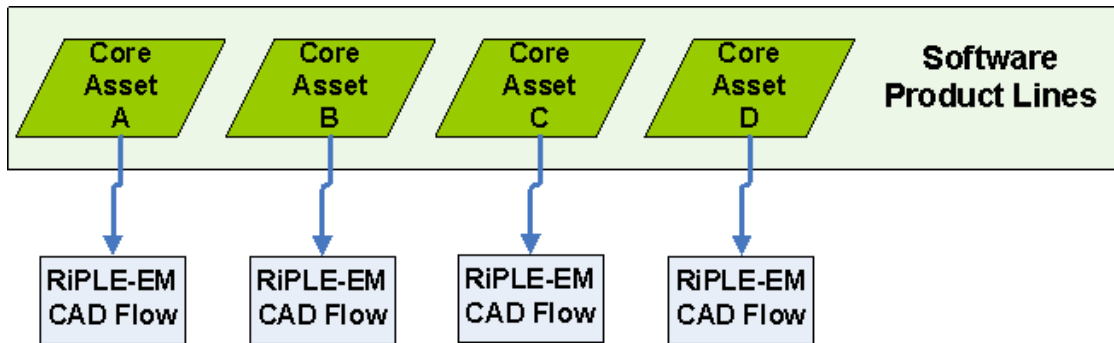
**Build Management.** The build management procedures and tools are responsible to guarantee the reproducibility of the generated build, and goes beyond that, by defining build strategies, defining exactly what part of the system is targeted to build (Bay, 1999). Generally it exists some kind of configuration or parameterization (conditional compilation directives, configuration files, aspect orientation, simply if-else blocks, etc.), to address the multiple variants of certain assets (Gacek and Anastasopoulos, 2001). One could relate build management with quality assurance, which was not the focus of this research. Furthermore, the build management discussed in this research includes not only the selection of the variants to be built, but includes the strategies and procedures of the build itself (such as tools, steps, etc.).

**Release Management.** Release management is the process through which software is made available to, and obtained by, its users (van der Hoek *et al.*, 1997). It is an important part of the overall software development process. In SPL, two levels of releases can be identified: *core asset releases* (targeted to the product development team) and *product releases* (generally targeted to market or end users). Each core asset can be developed and released independently from each other, since they are generally (re)used in more than one product, and thus, need to have specific activities to support the release procedures. The composition of the core assets releases may compose products that need to be released as well following specific procedures.

### 4.3 RiPLE-EM Usage Scenarios

In order to better understand the application of the RiPLE-EM in a real product line scenario, there are 3 (three) different examples scenarios of its usage. Before understanding the scenarios, it is important to have in mind that the RiPLE-EM is a release oriented process model, and therefore one cycle in this process model represents the planning, evolution, and release of a certain asset or product.

The *first scenario* is represented by Figure 4.3, and shows a fictitious product line in a given moment, where the product line focus is to develop core assets, thus, 4 (four) core assets are being developed. For the proper management of the evolution, each core asset development (release) follows the RiPLE-EM specific flow, in this case the RiPLE-EM CAD. Note that in this scenario there are no products being developed, only core assets, which is a typical scenario in the first phases of a product line adoption, where the core asset base will be constructed in order to be reused in the future products.



**Figure 4.3** RiPLE-EM usage example

The *second scenario* is represented by Figure 4.4 where the product line is in an opposite state from the first scenario. In this scenario, there are no core assets being developed in the moment, on the other hand, 4 (four) products are being developed, and therefore reusing existing core assets of the core asset base. Each product is handled separately, and for each product a new RiPLE-EM PD flow is started and followed.

The *third scenario*, and most common, is the scenario where both core assets and products are both being developed at the same time. In this scenario, represented by Figure 4.5, we have products being developed which are constrained by the core assets development (e.g. a product that depends on a core asset being developed). In this case, RiPLE-EM will be used both the RiPLE-EM CAD flow and the RiPLE-PD flow, one for each core asset or product release.

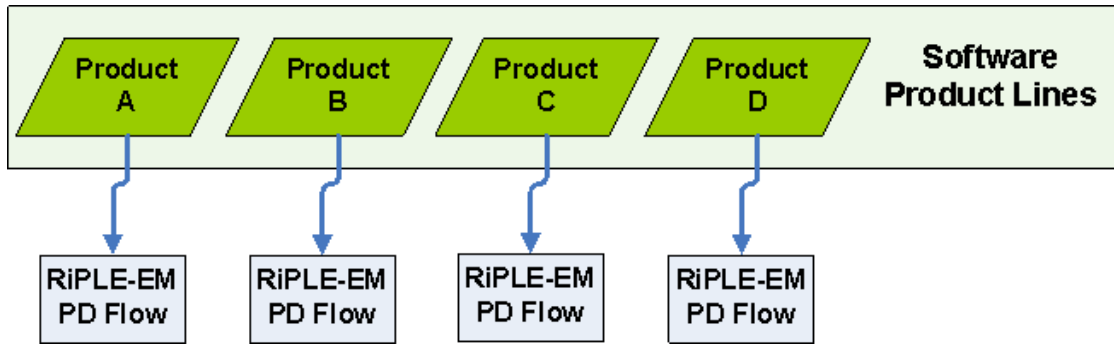


Figure 4.4 RiPLE-EM usage example

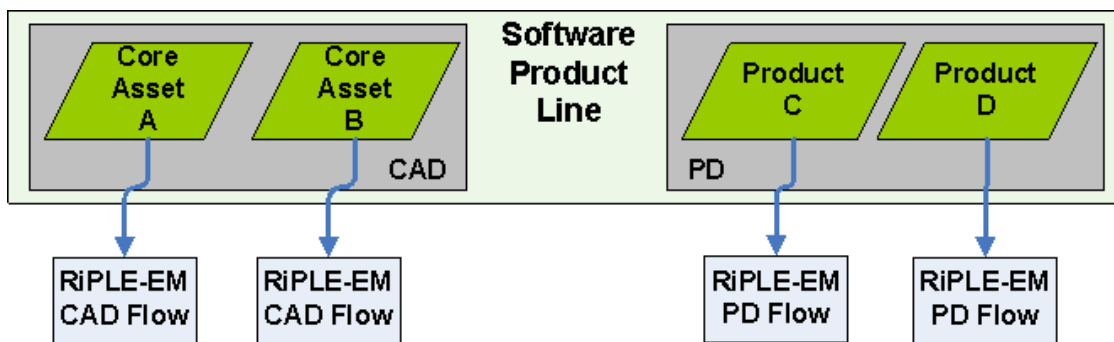


Figure 4.5 RiPLE-EM usage example

## 4.4 RiPLE-EM Core Asset Development x Product Development

As mentioned before, RiPLE-EM can be divided in two different flows, the RiPLE-EM for Core Asset Development (RiPLE-EM CAD) and the RiPLE-EM for Product Development (RiPLE-EM PD). These two different flows follow similar macro workflows, but the activities in the inner workflows have different goals.

RiPLE-EM CAD focuses on the proper evolution and release of a particular asset, including the identification of items and the release planning of a certain core asset, change management of those items, build management, and release execution practices where the core asset is made available for the product development.

The focus of RiPLE-PD is different, from the release planning, where assets to be included in the products are identified and tracked; the change management, where assets can be added or instantiated; build management, where the product is built, with all variability decisions resolved; until the release execution where the product is released

and published and the changes made in the release can be propagated back to core assets base.

In order to clarify the differences from RiPLE-EM for CAD and RiPLE-EM for PD, Table 4.1 summarizes some of the activities by expliciting the differences between the two flows.

	RiPLE-EM	
	RiPLE-EM CAD	RiPLE-EM PD
<b>Change Management</b>	<ul style="list-style-type: none"> <li>- Raise change request and propagation request</li> <li>- Process PR (raised in product development)</li> <li>- Creation of new core assets</li> <li>- Change specific asset</li> </ul>	<ul style="list-style-type: none"> <li>- Raise change request and propagation request</li> <li>- Process PR (raised in core asset development)</li> <li>- Create core asset instances</li> <li>- Change specific asset/instance</li> </ul>
<b>Build Management</b>	<ul style="list-style-type: none"> <li>- Resolve variability (partially)</li> <li>- Build core asset (private, integration, release build)</li> </ul>	<ul style="list-style-type: none"> <li>- All variability should be resolved.</li> <li>- Build product (private, integration, and release build)</li> </ul>
<b>Release Management</b>	<ul style="list-style-type: none"> <li>- Definition of supported products</li> <li>- Release schedule generation</li> <li>- Core asset release publishment (to product development)</li> </ul>	<ul style="list-style-type: none"> <li>- Identification of all core assets (and their versions)</li> <li>- Request a change on a core asset</li> <li>- Release schedule generation and monitoring (in case of core assets still being created/changed)</li> <li>- Product release publishment (to market/client/etc)</li> </ul>

**Table 4.1** RiPLE-EM CAD and RiPLE-EM PD Main Differences

In the *Change Management* discipline, the main difference is the creation of a core asset (in the CAD level) and the creation of a core asset instance (in the PD level). Besides that, the propagation request is also handled in both levels. The propagation request (PR) is the tool used to control the propagation of changes between core asset and product development.

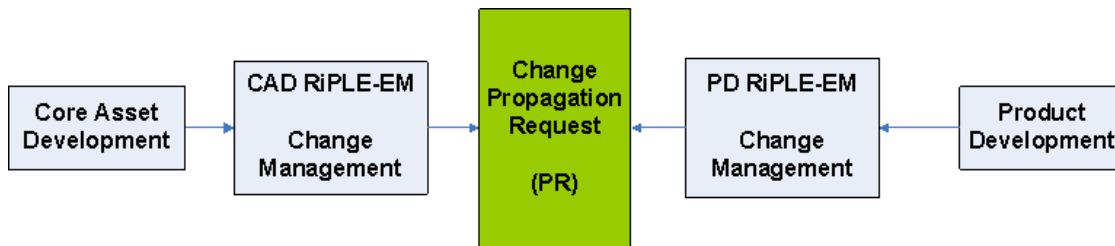
In the *Build Management* discipline, the difference between CAD and PD is the scope of the variability resolution (in case of compilation time variability). In the CAD level, most of the variability points may be still opened, or resolved partially, leaving some compilation variability decisions opened, to be resolved in the PD level. However, the build of a product should leave no variability decision opened.

In the *Release Management* discipline, which comprises the release planning and execution, the main difference resides in the release planning. In the PD level, the release planning must take in consideration the development of the core assets, and this development should be monitored in order to keep the product release planning up-to-date with the core asset development.

#### 4.4.1 RiPLE-EM CAD X PD Communication

Even though RiPLE-EM CAD and RiPLE-EM PD have different focus, since one is focused on the core asset evolution management and the other on the product evolution





**Figure 4.6** RiPLE-EM Change Propagation Request

management, they have a connection point represented in RiPLE-EM as a *Propagation Request*. The Propagation Request (PR) is a way to propagate the evolution (changes made to an asset or product) of an asset or product to another asset or product.

Figure 4.6 shows the propagation request serving as a bridge of communication between the core asset development and product development changes.

## 4.5 RiPLE-EM Support Tools

The support tools used to help the application of the process are described in this Section, divided by discipline.

RiPLE-EM is designed to be independent from the tools used by each organization, so any kind of tool that offer support towards the RiPLE-EM disciplines can be effectively used.

- **Change Management Tools.** To manage changes and track bugs, change requests and propagation requests any existing bug tracking tool can be useful. The most common ones and open-source (since open-source tools are generally more easy to introduce due to financial constraints) are: TRAC<sup>1</sup>, Mantis<sup>2</sup> and Bugzilla<sup>3</sup>. To manage changes, and help analysing the impact of the changes in the software, any kind of traceability support is important, since traceability matrix in a spreadsheet document (no automated) until automated tools to track the tracelinks among artifacts.
- **Build Management Tools.** Any build automation tool such as the open source

---

<sup>1</sup>Trac project home - <http://trac.edgewall.org/>

<sup>2</sup>Mantis But Tracking project home - <http://www.mantisbt.org/>

<sup>3</sup>Bugzilla project home - <http://www.bugzilla.org>

ANT<sup>4</sup> and MAVEN<sup>5</sup>, or even proprietary build automation tool such as MSBuild<sup>6</sup>. Besides these general build automation tools, there are some SPL specific tools which help in the build automation and generation in SPL contexts such as gears (Krueger, 2008) and pure::variants (Beuche, 2008).

- **Release Management Tools.** The main need of the release management discipline is a tool to manage the releases in a central repository so it can be accessible by other teams in the same product line. Any tool offering support can be used, and version control can also be used to manage the distribution (releases) of components and products.

Besides these specific tools for each discipline, there are also ALM tools that covers most RiPLE-EM disciplines and a set of other disciplines. As an example there are the IBM Rational Team Concert (RTC)<sup>7</sup> and the MS Visual Studio Team System (VSTS)<sup>8</sup>.

## 4.6 RiPLE-EM Related Roles

The roles used specifically in RiPLE-EM are listed and described next:

- **Build and Release Engineer:** Is the role responsible for building and releasing products and core assets, this role must have knowledge of build techniques, variability management, and must have sufficient access to publish release and builds for tests. This role may also be named as *Software Configuration Management Engineer* (as it is more usual in certain contexts).
- **CCB:** This role is a fundamental role in software evolution. The CCB is the group (or individual) responsible for the analysis of both change and propagation requests. All requests have to be analyzed by the CCB, and the CCB is the responsible for approving the change or the propagation request. The group composition is flexible and depending on the change or propagation being requested the CCB can have different members.

Since the RiPLE process concerns not only Evolution Management, but all other phases in the SPL engineering field (such as scoping, requirements, implementation,

---

<sup>4</sup>Ant project home - <http://ant.apache.org/>

<sup>5</sup>Ant project home - <http://maven.apache.org/>

<sup>6</sup>MSBuild General Reference - <http://msdn.microsoft.com/en-us/library/0k6kkbsd.aspx>

<sup>7</sup>IBM Rational Team Concert - <http://www-01.ibm.com/software/awdtools/rtc/>

<sup>8</sup>MS Visual Studio Team System - <http://msdn.microsoft.com/en-us/teamssystem/default.aspx>

testing), there are some roles related to some of these other phases that are also present in RiPLE-EM, but are not specific of it. These other roles are also described next:

- **Core Asset Manager:** The Core Asset Manager is responsible for the development of a core asset. Activities such as planning the release and maintain the release schedule is this role's responsibility.
- **Product Manager:** This role is responsible for the product management, all product management activities are performed by this role.
- **SPL Engineer:** This role comprises technical engineering stuff such as the *SPL Architect*, *SPL Developer*.
- **Any Role:** As the title says, this represents a role that is related to every role in the process. If a certain activity is assigned to *Any Role*, it means that any of the roles in the process may be able to execute the activity.

The roles associated with each RiPLE-EM activity are described in Tables 4.2, 4.3 and 4.4.

In small contexts, the roles of *Core Asset Manager* and *Product Manager* can be merged into only one role (some sort of a *SPL Manager*), which is the role responsible for both core asset and product management.

It is important to notice that the roles described in this Section are a sub-set of RiPLE roles (the more general process).

## 4.7 RiPLE-EM Activities Summary

All RiPLE-EM CAD activities, detailed in Chapter 5, and RiPLE-EM PD activities, detailed in Chapter 6 are summarized in this Section according to the evolution management discipline.

Table 4.2 summarizes the change management discipline, Table 4.3 summarizes the build management discipline, and Table 4.4 summarizes the release management discipline.

Each Table contains the activities of the discipline, the activity development level applicability (if it is applicable to CAD, PD or both), the input and output work products, and the roles associated.

#### 4.7. RIPLE-EM ACTIVITIES SUMMARY

Change Management	Activity	Development Level	Input	Output	Role
	Request Change	CAD, PD	Not Applicable	- Change Request	Any Role
	Request Propagation	CAD, PD	Not Applicable	- Propagation Request	Any Role
	Delegate Changes	CAD, PD	- Change Request	- Change Request	CCB
	Analyze Change Request	CAD, PD	- Change Request	- Change Request	CCB
	Create Core Asset Infrastructure	CAD	- Change Request	- Asset Infrastructure	
	Change Asset Artifacts	CAD, PD	- Asset to be changed - Change Request	- Asset changed	SPL Engineer
	Analyze Propagation	CAD, PD	- Propagation Request	- Propagation Request	CCB
	Feedback to Requester	CAD, PD	- Propagation Request	- Propagation Request	CCB
	Integrate Feedback Changes	CAD	- Asset to be changed - Propagation Request	- Asset changed	SPL Engineer
	Create a New Asset Instance	PD	- Asset	- Asset Instance	SPL Engineer
	Rebase Asset	PD	- Asset to be changed - Propagation Request	- Asset changed	SPL Engineer

**Table 4.2** RiPLE-EM Change Management Activities Summary

Build Management	Activity	Development Level	Input	Output	Role
	Resolve Pending Variabilities	CAD, PD	- Asset - Release Map	- Asset without pending variability	Build and Release Engineer
	Private System Build	CAD, PD	- Build Script - Infrastructure to Build	- Component Build - Product Build	Build and Release Engineer
	Continuous Integration Build	CAD, PD	- Build Script - Infrastructure to Build	- Component Build - Product Build	Build and Release Engineer
	Release Build	CAD, PD	- Build Script - Infrastructure to Build	- Component Build - Product Build	Build and Release Engineer
	Verify Build	CAD, PD	- Build Log	Not Applicable	Build and Release Engineer
	Identify Variability Type	PD	- Asset - Release Map	Not Applicable	Build and Release Engineer

**Table 4.3** RiPLE-EM Build Management Activities Summary

## 4.8. RIPLE-EM WORK PRODUCTS

Release Management	Activity	Development Level	Input	Output	Role
	Define Supported Products	CAD	- SPL Roadmap	- Release Notes Draft	- SPL Manager - Build and Release Engineer
	Generate Release Schedule	CAD, PD	- Asset/Product Map	- Release Schedule	- SPL Manager - Build and Release Engineer
	Identify Deliverables	CAD, PD	- Asset/Product Infrastructure	- Release Notes Draft	- SPL Manager - Build and Release Engineer
	Update Release Map and Schedule	CAD, PD	- Release Schedule	- Release Schedule	- SPL Manager - Build and Release Engineer
	Identify All assets version and availability	PD	- SPL Roadmap - SPL Feature Model	- Product Release Map	- SPL Manager - Build and Release Engineer
	Request New Asset Version	PD	Not Applicable	- Change Request	Any Role
	Monitor and Track Assets Development	PD	- Release Schedule	- Release Schedule - Product Release Map	- SPL Manager - Build and Release Engineer
	Gather Release Information	CAD, PD	- Product Release Map - Release Schedule	- Release Notes Draft	- Build and Release Engineer
	Consolidate Release Notes	CAD, PD	- Release Notes Draft	- Release Notes	- SPL Manager - Build and Release Engineer
	Publish Release	CAD, PD	- Release Package	- Published Release Package	- Build and Release Engineer

**Table 4.4** RiPLE-EM Release Management Activities Summary

## 4.8 RiPLE-EM Work Products

The work products listed in Section 4.7, as input and output to RiPLE-EM activities are general SPL work products, however, there are some of them that are proposed by RiPLE-EM and may not exist in regular SPL environments, thus, they are described as follows:

- **Change Request.** A change request is generally represented by a ticket in a bug tracking system, or a issue tracker. This ticket can be used to request a change in an existing asset, and can serve as a task request (e.g. creation of a new asset).
- **Propagation Request.** The Propagation Request (PR) is a way to propagate the evolution (changes made to an asset or product) of an asset or product to another asset or product. Propagation request serves as a bridge of communication between the core asset development and product development changes, as described in Section 4.4.1.
- **Release Map.** The release map is a matrix of all core assets used in a certain product. It can be a simple matrix in a spreadsheet, but it can also be automated

by some tool, helping the visualization of dependency and impacts (the impact of changing a core asset for example).

- **Release Notes.** The release Notes is the artifact that describes all information about a certain product/core asset release. All information related to the release should be inside of the Release Notes.
- **Build Script.** The build script is the script responsible for generating the build of a core asset or a product. The script contains all steps necessary to compile and generate the executable/binary of a core asset or a product. Other types of builds that does not use build scripts necessarily should replace this build script work product.

The available templates and checklists are available in Appendix B.

## 4.9 Eclipse Process Framework

The RiPLE-EM process was modeled inside the Eclipse Process Framework (EPF)<sup>9</sup>, which aims at providing an extensible framework and exemplary tools for software process engineering - method and process authoring, library management, configuring and publishing a process.

EPF uses the Software Process Engineering Meta-Model (SPEM), that defines a formal language for describing development processes. EPF is based on SPEM 2.0, released on April, 2008 (SPEM, OMG, 2008).

Since, one of the main goals of EPF is to provide the reuse among sets of reusable activities (called *Method contents*), the EPF structure is divided into two main categories.

- **Method Content:** A set of defined tasks flow, roles, artifacts and guides to accomplish some goal.
- **Processes:** Process flows which consumes the method contents, reusing the previously defined activities.

### 4.9.1 Method Content and Processes

Both method contents, and processes are divided into some concepts, according to the SPEM (SPEM, OMG, 2008).

---

<sup>9</sup>Eclipse Process Framework web site - <http://www.eclipse.org/epf/>

The Method Content contains the following concepts:

- **Role:** Roles define a set of related skills, competencies and responsibilities. Roles perform tasks.
- **Work Product:** Work Products (in most cases) represent the tangible things used, modified or produced by a Task.
- **Tasks:** A Task defines an assignable unit of work (usually a few hours to a few days in length).
- **Guidance:** Guidance may be associate with Roles, Tasks, and Work Products, and may have the form of a checklist, an example, a template and etc.

The Processes contains the following concepts:

- **Capability Pattern:** Capability Patterns define the sequence of related Tasks, performed to achieve a greater purpose.
- **Delivery Process:** Defined using Work Breakdown Structures and/or Activity Diagrams. Defines end-end full-lifecycle process and may include iterations, phases, milestones.

## 4.9.2 Benefits

The main benefits from using EPF are:

- **Reuse:** The method contents can be reused throughout the processes.
- **Web Site generation:** EPF generates automatically a web site containing all information of the modeled process, making the publication of the process a very easy task.

## 4.10 Chapter Summary

The RiPLE-EM process model is part of a more general process called **RiSE Product Line Engineering (RiPLE)**, which concerns with the software life-cycle for software product lines. In this context, RiPLE-EM concerns with the evolution management of software product lines, or, more specifically, the evolution management of the assets and

products inside a product line. RiPLE-EM involves three evolution disciplines: change management, build management and release management.

RiPLE-EM is a process to guide evolution on both SPL development levels (Core Asset Development and Product Development) and it has the concept of Propagation Requests, which is the communication point between core asset and product development.

The RiPLE-EM specific roles are the *Build and Release Engineer*, responsible for building core assets and products and releasing, and the *CCB* which is the group responsible to analyze both change and propagation requests.

Next Chapter presents RiPLE-EM specific workflows, activities and steps for the evolution management of the core asset development level of software product lines.



*“We are what we believe we are.”*

C. S. Lewis

# 5

## RiPLE-EM CAD

The RiPLE-EM for CAD defines workflows, activities, steps, roles and work products to properly control the evolution and changes of the core assets throughout their life-cycle.

Thus, this Chapter presents in a detailed level, including all flows, activities and steps of the RiPLE-EM for Core Asset Development. Section 5.1 describes the RiPLE-EM main flow for core asset development. Section 5.2 discusses the release planning activities, Section 5.3 discusses the change management activities, Section 5.4 presents the build management activities and Section 5.5 details the Release execution activities. The Chapter summary is presented in Section 5.6.

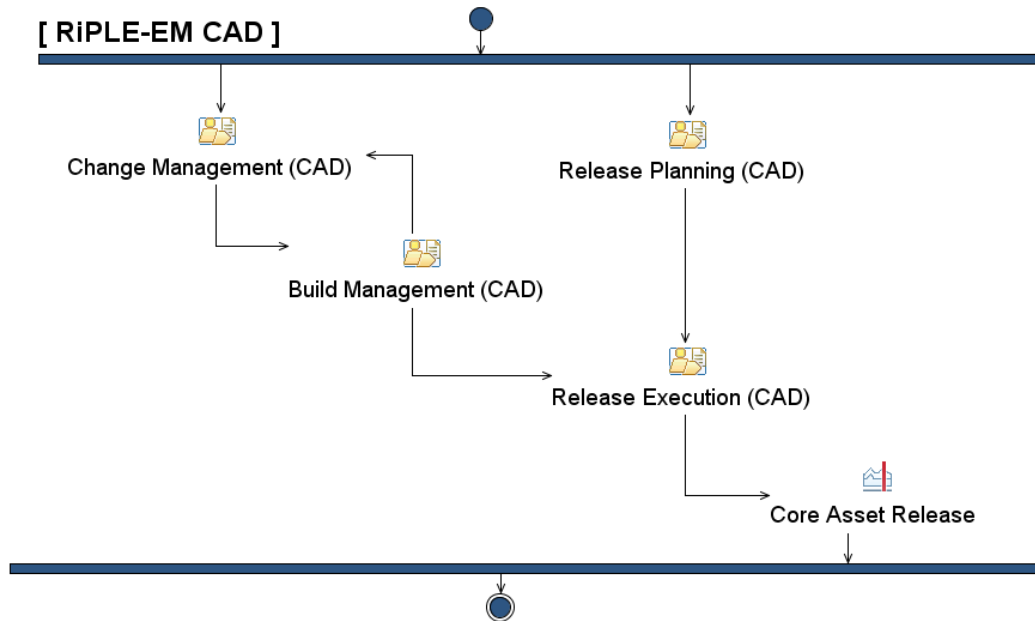
### 5.1 RiPLE-EM for Core Asset Development

The focus of RiPLE-EM for CAD activities is to guarantee the proper evolution of core assets, from the moment of their planning and creation, through its evolution, until the moment the core asset is released for product development.

As stated before, RiPLE-EM is a release oriented process model, thus, each core asset development will finish with its release for the product development team to (re)use it according to Figure 5.1 .

RiPLE-EM comprises the whole cycle from the core asset release planning, until the release execution of the proper core asset. The RiPLE-EM CAD cycle is represented by Figure 5.1

Each activity in Figure 5.1 is expanded into several tasks to achieve specific goals, and each activity will be detailed in the following Sections.



**Figure 5.1** RIPLE-EM CAD Macro Flow

## 5.2 RIPLE-EM CAD - Release Planning

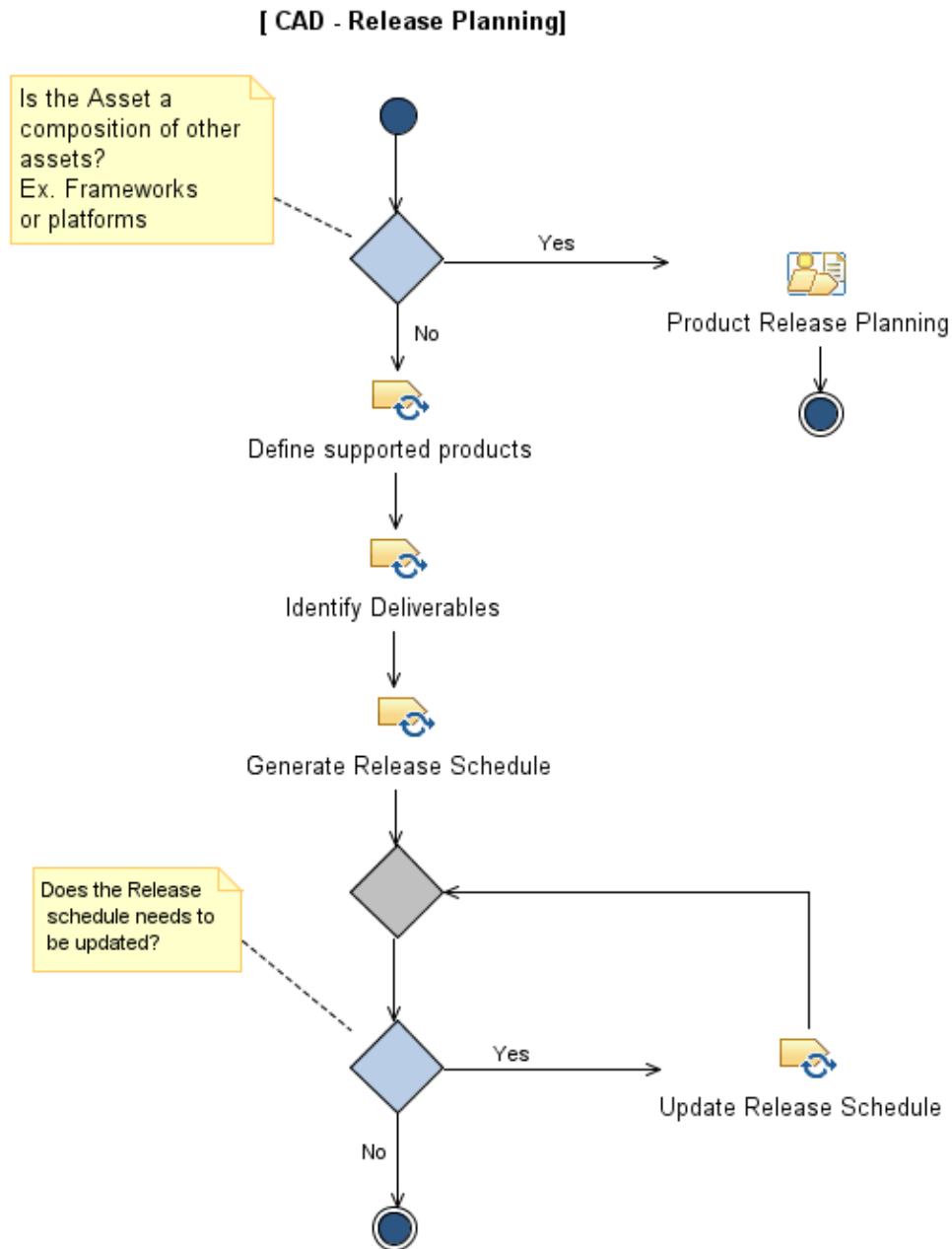
The activity of planning the release of a certain core asset is performed in parallel with the development, and it can be refined until the release execution moment.

The activity flow to accomplish the release planning is shown in Figure 5.2. The activities inside this flow are described next:

- **Define Supported Products** The proposal here is to start writing the release notes from the moment the release planning starts, so all information can be stored in this draft release notes until the release execution (when the release notes will be consolidated).

One important information about the release of a core asset is the products that this core asset supports. This information can be obtained by verifying all products inside the SPL portfolio and identifying the ones that the core asset can support. This activity is composed of the following steps.

Steps:

**Figure 5.2** RiPLE-EM CAD: Release Planning Flow

- **Create a draft release notes for this core asset release:** Draft Release notes creation to start updating information on it.
- **Define what products the core asset supports:** Define what products of the product line, the core asset supports and can be used in. This information shall be updated in the draft release notes.
- **Identify Deliverables** Identify all items that will be part of the release package/distribution. Generally, there are at least two possibilities of deliverables packaging. The first possibility is to deliver only the binaries/executables(jar files, dll's, etc.), and the second possibility is to deliver the source code and related documentation (requirements, production plan, test cases, etc.) along with the binary/executable.

It is important that the package creation is automatic, so it essential to know the parts of the release package in advance to automate it. This activity is composed of the following steps.

Steps:

- **Identify Items to be included in release package:** All files that the release package will contain, need to be identified and documented in the draft release notes. This information is useful to prepare the build script packaging automation. This information can be updated in the draft release notes.
- **Generate Release Schedule** Generate Release Schedule for the core asset, comprising the major milestones. No release schedule template is proposed by RiPLE-EM in order to increase flexibility, since generally the release schedule is part of the macro core asset development schedule.
  - **Establish release milestones and Generate Release Schedule:** In this step, the release schedule is created taking in consideration all development constraints and stakeholder needs.
- **Update Release Schedule** Since some products (on the product level) depend on this core asset release, the schedule needs to be constantly updated, and reflect the real situation of the development. Products in the product line may base their releases on the core asset releases they depend on. Based on that, it is very important that the core asset release schedule is constantly updated to reflect every change in the major release milestone.

- **Synchronize core asset schedule with any product release and Update Core Asset Release Schedule:** Since some products depends on core assets release, it is always important to follow these product release schedules to see if the core asset release schedule is accordingly.
- **Notify product managers about the core asset release schedule change:** All changes to the core asset release schedule that will have any impact on the products releases must be notified to product managers, so they can re-plan their product release schedules and maps.

## 5.3 RiPLE-EM CAD - Change Management

When the development of the core asset takes place, in other words, the creation of the core asset and its evolution (changes), the change management flow starts.

The change management flow is the flow which interfaces with the implementation (realization) of the software product line, where the changes really occur, assets are created, and modified (evolved).

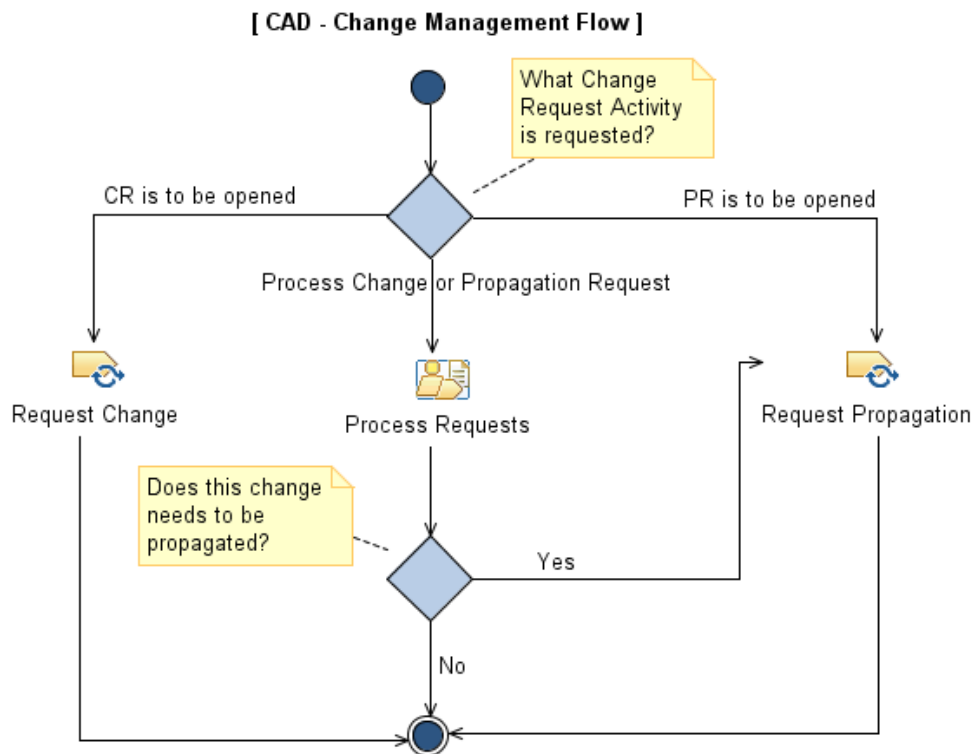
In order to control the changes done to the core assets, the flow in Figure 5.3 is proposed. The activities inside this flow are described next.

- **Request Change** Whenever a change needs to be done to an asset, the formal change process must be followed. This formal process starts with the requisition of the change.

The Change Request (CR) must contain all important and detailed information about the change being requested in order to the CCB group analyze the change being proposed and then approve it for the actual implementation of the change. This activity is composed of the following steps.

Steps:

- **Complete the Change Request form:** Every information needed should be provided in details. The more detailed the information is, the easier it is for the CCB group to analyze. It is important that the information provided is clear. Some of the provided information may be changed after the analysis of the CR  
( e.g. Bug fix request - The requester informs that the bug is in module X, and after the analysis the CCB find out that it is in module Y, so the requester original information is changed by the CCB. )



**Figure 5.3** RiPLE-EM CAD: Change Management Flow

- **Revise form and submit:** Before submitting the CR, make sure that all information are consistent and clear.
- **Request Propagation** The Propagation Request(PR) plays an important role in this process, being the main gateway for changes to be propagated from Core Asset Development to Product Development and vice-versa. The PR is further described in Section 4.4.1.

Whenever a core asset or product release is published (or in any other time), a PR could be submitted to update the other development level. For example, if a new core asset release is published, a new propagation request can be opened for the product that uses that core asset to update the version it uses.

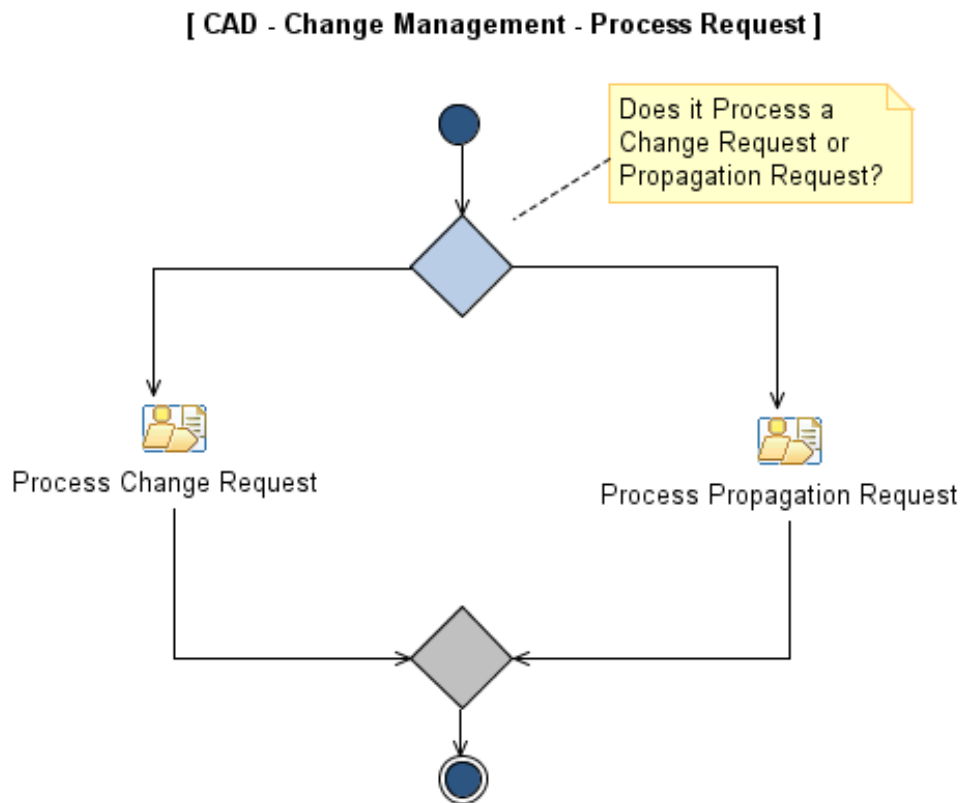
This process of propagation starts with the PR. The PR must contain all important and detailed information about the propagation being requested in order to the CCB group to analyze the PR and approve the propagation of that change/release.

Steps:

- **Define change set targeted to propagation:** In this step, the requester has to list the change set that will be propagated [ e.g. set of CRs (a release), or a single CR ]. In theory, every PR has to be related with one or more CRs.
- **Select Propagation Type in the Change Control Tool:** Propagation Requests can be of different types ([Anastasopoulos et al., 2009](#)):
  - \* **Feedback (From the PD -> CAD):** Where the changes in the PD level are propagated back to the CAD level, feed backing the CAD with those changes, turning those changes reusable by other products, since it is on the core asset.
  - \* **Rebase (From the CAD -> CAD):** It occurs mainly when a new version of a core asset is released and the products that uses this core asset can rebase their instances (either completely or partially by merging only the differences).

The other possibility is the internal propagation ([Atkinson et al., 2002](#)):

- \* **Internal product Development propagation (From the PD -> PD):** Where the change is propagated from one product to another one (or a set of product).



**Figure 5.4** RiPLE-EM CAD: Process Request Flow

\* **Internal Core Asset Development propagation (From the CAD -> CAD):** Where the changes are propagated from one core asset to another core asset (or a set of core assets).

– **Revise Form and Submit:** Before submitting the PR, make sure that all information are consistent and clear.

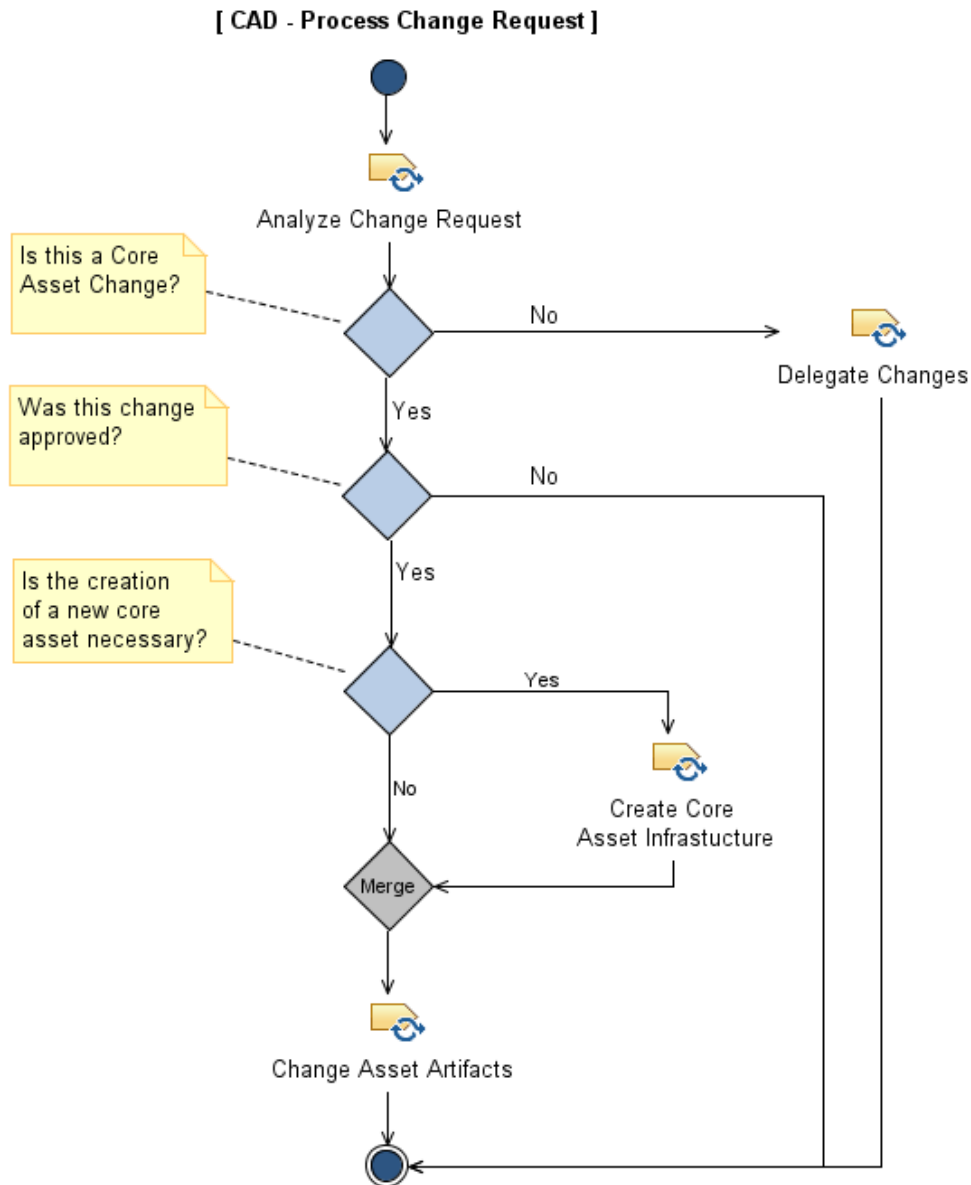
Besides requesting changes and propagation, the change management flow supports changes and propagation processing, by expanding the *Process Requests* activity in Figure 5.3. When this activity is expanded, a new flow is started, represented by Figure 5.4.

In Figure 5.4, we can see two other activities: (1) *Process Change Request*, and (2) *Process Propagation Request*.

Expanding (1), we have another flow, as we can see in Figure 5.5, and the activities are detailed next.

- **Analyze Change Request:** Every change requested must be analyzed in order



**Figure 5.5** RiPLE-EM CAD: Process Change Request Flow

to avoid problems related to the change. The checks that must be covered are described in the Change Request Checklist, in Appendix B, and all topics should be analyzed in order to approve the implementation of a specific CR. This activity is composed of the following steps.

Steps:

- **Identify if the CR is related to CAD or PD:** It is important to verify if the change requested relates to CAD or PD.
  - eg.1** If this activity is being performed in the CAD, then all CRs analyzed should be related to CAD, otherwise the CR should be forwarded to PD, and there be properly managed.
  - eg.2** If this activity is being performed in the PD, then all CRs analyzed should be related to PD, otherwise the CR should be forwarded to CAD, and there be properly managed.
- **[optional] Identify if the problem/bug is reproducible:** If the CR is a bug report, then it is necessary to verify if the bug can be reproduced. If the problem is not reproducible, this has to be documented in the CCB analysis decision.
- **Identify all change's impacts and feasibility:** This analysis has to follow the attached checklist (CR Analysis checklist) and each check have to be checked in order to assess all impacts of the change proposed.
- **Negotiate and Assign a person to implement the CR:** The CR resolution must be assigned to someone in the team. This person would be responsible for the implementation(realization) of this CR.
- **Document the CCB Analysis:** The result of the analysis must be documented inside the CR, following the proposed "CCB Analysis documentation template" available in Appendix B.
- **Delegate Changes:** This activity reflects the responsibility delegation of the CR from Core Asset to Product Development or vice-versa. This activity is composed of the following steps.

Steps:

- **Change the Development Level in the CR:** The development level (core asset development or product development) of the change must be changed,

characterizing that the change will not be treated in the level it was analyzed, and then, the delegated level has to analyze and resolve the CR.

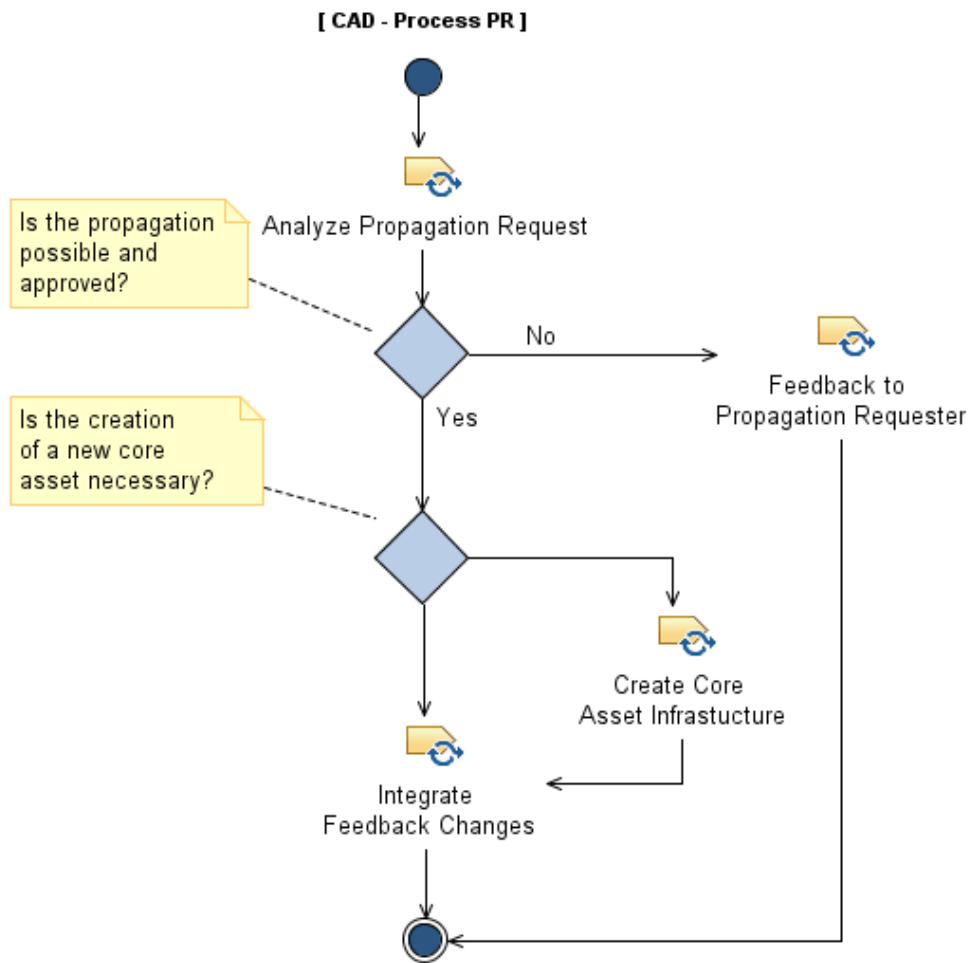
- **Create Core Asset Infrastructure:** This activity reflects the creation of a new core asset infrastructure (repository folders structure, change control tool organization, etc.), according to the analysis previously made. The infrastructure of the core asset must be created, the repository structure for the core asset also, and change control tool for the core asset must be configured. This activity is composed of the following steps.

Steps:

- **Create core asset directory structure locally:** The core asset and its directory structure must be created locally (in the engineer machine), and only after in a later activity it is sent to the repository. If there is already an organizational standard for directory creation, it should be followed.
  - **Create a specific project in the Change Control Tools:** A new project space must be created in the change control tool to store the CR and PR related to the new core asset. This area in the change control tool must be created and must include information such as asset's modules, components, etc.
- **Change Asset Artifacts:** The purpose of this activity is to change existing SPL assets for any development purpose. This activity encapsulates all techniques, methods and paradigms of assets implementation. Under this activity, all existing assets are properly changed, according to the the analysis of the CR performed by the CCB. This activity applies also if the modification requires some changes in variation points (e.g. addition, removal of variation points). All changes necessary should be done under this activity, and even other implementation process or method can be plugged-in in this point. The way the assets are going to be changed is not the focus of this process, and then it will not be detailed here. This activity is composed of the following steps.

Steps:

- **Realize the changes associated with the CR:** All changes to existing asset's artifacts must be done in this activity.
- **[optional] Perform unit tests:** To assure the correctness of the changes in the unit level, some unit tests may be applicable (automatically or not).



**Figure 5.6** RiPLE-EM CAD:Process Propagation Request Flow

- **[optional] Perform smoke tests:** To minimally assure the correct behavior of the asset being change, smoke tests (Berczuk and Appleton, 2002) can be performed.

Back in Figure 5.4, by expanding (2) *Process PR*, we have another flow, represented in Figure 5.6 and the activities are described next. The activities presented in Figure 5.6 are described next.

The activity *Create Core Asset Infrastructure* is not described in the following since it was previously described (as part of Figure 5.5).

- **Analyze Propagation Request:** This activity purpose is to analyze the feasibility of propagating the changes among variants/core asset. Every PR must be analyzed

in order to guarantee the correct propagation of the changes. The checks described in the *PR Analysis checklist*, described in Appendix B, must be analyzed in order to approve the realization of a specific PR. The result of the analysis must be documented inside the PR. This activity is composed of the following steps.

Steps:

- **Identify "from" where the changes are being propagated:** Verify if the "from" version of a PR is compatible with the core asset receiving the change.
- **Identify the propagation type:** Two types of change propagation: (i) *Complete Replacement*, where the change set completely replaces the old version. In this type of propagation, much attention should be put into compatibility problems, and classical component upgrade problems (Voas, 1998), in order not to alter the right behavior of the system; and (ii) *Merge versions*, where the whole change set, or just a part of it, is merged against the already existing target version. In this case much attention should be put in merge conflicts resolution.
- **[Optional] Isolate the changes to be propagated:** This step is only applicable in the Merge propagation type. In this case, it has to be identified what is the change set (or part of it) that needs to be merged.
- **Document the decision to propagate:** The result of the analysis must be documented inside the PR, following the proposed "CCB Analysis documentation template".
- **Feedback to Propagation Requester:** The purpose of this activity is to inform the propagation requester the reasons the propagation requested is not approved. This activity is composed of the following steps.

Steps:

- **Document the reasons the PR was not approved:** In order to document, and serve as a communication channel between the CAD and PD, the decisions regarding the PR must be documented inside the PR.
- **Integrate Feedback Changes:** Sometimes, the changes made in the product instance of a core asset, or even a product specific asset must be propagated to CAD to integrate the changes back into the core asset or create a new core asset in the core asset base. This activity is composed of the following steps.

Steps:

- **Propagate changes from PD to CAD:** This activity is the actual propagation of changes and must follow the propagation type defined in the propagation analysis.
- **[optional] Resolve risen conflicts:** Only performed if the Propagation type is Merge. All risen conflicts must be resolved in order to promote changes to a build to be tested.

## 5.4 RiPLE-EM CAD - Build Management

When the changes proposed are complete, a build of the core asset should be created for different purposes (for developer testing, for integration testing or for releasing).

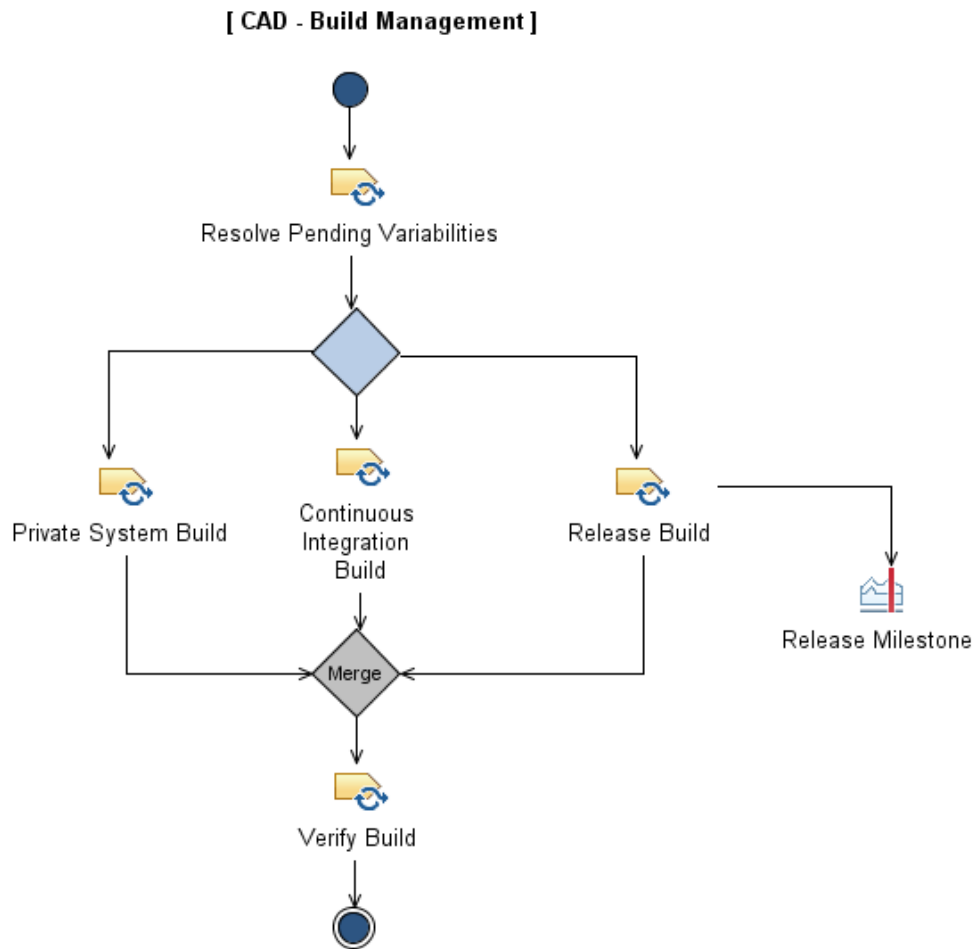
It is important to observe that some existing tools (such as gears ([Krueger, 2008](#)) and pure::variants ([Beuche, 2008](#))) in the SPL build management area already substitute (automatically) most of the activities in this flow. Thus, this flow can be replaced by appropriate and automatic tools.

RiPLE-EM activity flow to guide build activities is illustrated in Figure [5.7](#). All activities in this flow are explained next.

- **Resolve Pending Variability:** All variability that were not resolved (mainly variabilities with compilation time binding) needs to be identified and resolved in order to launch the build. When variability is resolved, the decisions have to be reflected in the build command, for the proper build execution. This activity is composed of the following steps.

Steps:

- **[optional]All Conditional compilation variability is identified and resolved:** All Conditional compilation variability is identified and resolved before launching the build.
- **Private System Build:** The Private System Build is used by the individual developer working on an isolated development task to verify the consistency and correctness of their changes before committing them to the repository where they will be visible to the rest of the development team. The developer creates a Private Workspace and populates it with the latest working versions from the code line in order to begin work on their development task. After making changes in their



**Figure 5.7** RiPLE-EM CAD: Build Management Flow

Private Workspace, the developer needs to ensure that the source changes work correctly and will not "break the build" for the rest of the team. This activity is composed of the following steps.

Steps:

- **Define the build completeness:** Define the build completeness: - Clean full build: The environment should be totally cleaned before the build execution. - Incremental build: Other parts from previous build can be reused (It is faster, but less secure)
  - **Assure build environment is similar to a release build environment:** The purpose of having a build environment equals (or even much similar) to release build environment is to avoid future problems in the release generation; and
  - **Launch Build:** Launch the build of the core asset. Depending of the technology being used, the launch can be done in several ways. Some typical ways are using the command line instructions, building inside the used IDE, or simply do not build (in case of interpreted languages such as PHP or any other).
- **Continuous Integration Build:** The Continuous Integration Build is generally generated automatically (or even manually) to incorporate the most recently completed development tasks into a single, stable, consistent build.

An integration build is preferably a full build (if time and bandwidth allows) whose results visibly impact the progress and coordination of the entire development team and all the active development tasks at that time. The frequency of an integration build may be as often as one per development task (at the end of each task) or may be every few tasks (or even daily/nightly). In some cases, there may be two "flavors" of integration builds: one that is done after every development task and does only an incremental build (and possibly runs only a subset of tests), and one that runs daily or nightly and does a full build and runs a more complete set of tests. In either case, the purpose of the integration build is to centrally coordinate and synchronize all ongoing work in small bits and pieces to avoid "big bang integration" at the end of a release/iteration, and to provide a regular rhythm for the development team to tackle integration issues early and aggressively, one small piece at a time. This activity is composed of the following steps.

Steps:



- **Define the build completeness:** Define the build completeness: - Clean full build: The environment should be totally cleaned before the build execution. - Incremental build: Other parts from previous build can be reused (It is faster, but less secure)
  - **[optional] Merge changes to one single code line:** Merge changes to one single code line (If changes are in different branches).
  - **Assure Build Environment is similar to Release Build Environment:** The purpose of having a build environment equals (or even much similar) to release build environment is to avoid future problems in the release generation.
  - **Launch Build:** Depending of the technology being used, the launch can be done in several ways. Some typical ways are using the command line instructions, building inside the used IDE, or simply do not build (in case of interpreted languages such as PHP or any other). Regarding the use of a continuous integration tool, there are mainly two build possibilities: *Automatically*, build launched and controlled by a continuous integration tool or *Manually*, build launched and controlled manually by engineers. In the case of the *Automatically* possibility, the next step is also executed automatically by most continuous integration tools.
  - **Apply Label on Repository build Configuration:** Apply Label on repository build configuration. The possibilities can be automatically (label applied by continuous integration tools) or manually label (applied manually by the *Build Engineer*).
- **Release Build:** The release build is the "formal" build package for the core asset. Some core assets may require delivery and distribution of their source code in order to serve as build input to other core assets; while other components may require delivery and distribution of only binaries, or binaries and interface definitions (e.g., header files in C/C++). This activity is composed of the following steps.

Steps:

- **Define the build completeness:** Define the build completeness: - Clean full build: The environment should be totally cleaned before the build execution. - Incremental build: Other parts from previous build can be reused (It is faster, but less secure)

- **Identify the Release Type:** It is suggested that the release type (beta, bug fix...) should be reflected in the release label version. E.g. PRODUCT-REL-01.00B or PRODUCT-REL-01.00C. This same release label should be a parameter to Release Build script, to be included in some manifest file in order to be possible to later identify given a certain package what version is it.
  - **Launch Build:** Launch Build using the command line with the resolved compile time variability if applicable.
  - **Apply Label on Repository build Configuration:** Apply Label on Repository build Configuration.
- **Verify Build:** After the build finishes, the logs and results from the build execution needs to be verified to identify if any error happened during the build generation.

Both the final result provided from the build automation tool, as well as the logs from the generation needs to be checked. This activity is composed of the following steps.

Steps:

- **Verify if the build result was successful:** This verification should not only comprise the build final result, but also the whole build log. The idea is search for any error during the execution that did not lead to a failure in the build execution.
- **Notify internal team of the build result (Not applicable for Private System Build):** After the confirmation that the build is really successful, all stakeholders should be notified of the build production

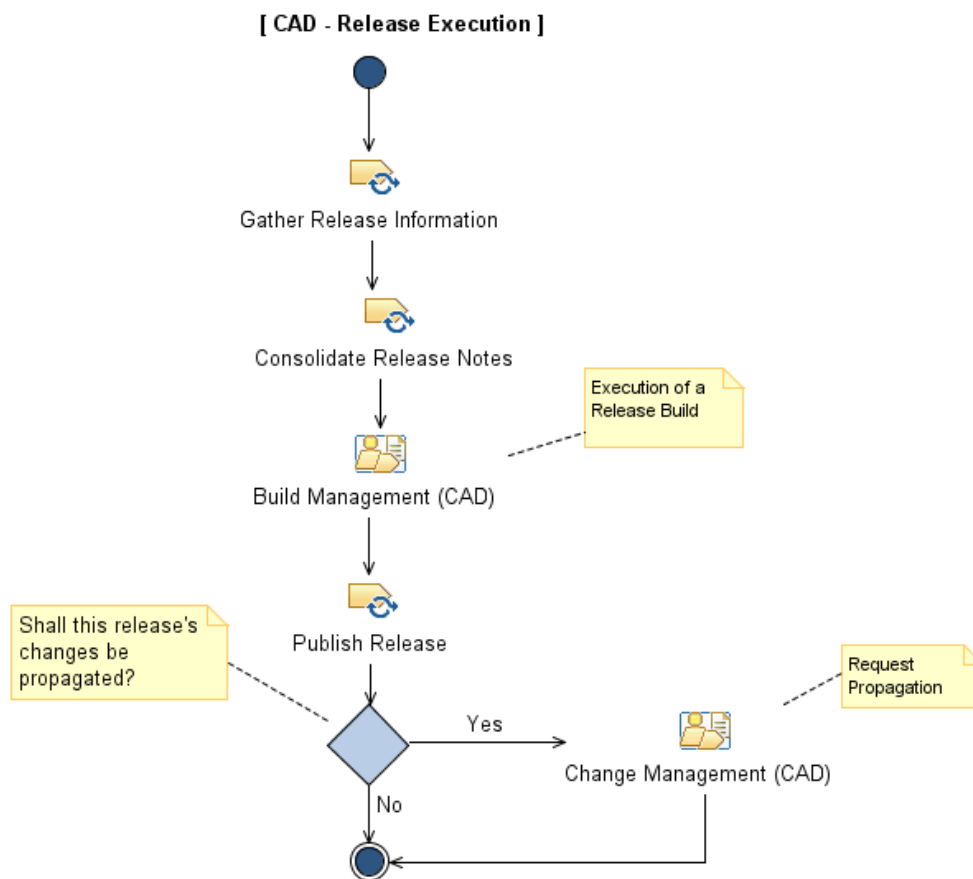
## 5.5 RiPLE-EM CAD - Release Execution

Once the build package is tested and approved for release (to the product development), the RiPLE-EM CAD Release Execution flow takes place.

RiPLE-EM CAD activity flow to guide release execution is illustrated in Figure 5.8. All activities in this flow are explained next.

- **Gather Release Information:** Gather Information about the Release being produced. Some information are already inside the draft release notes, but still, some are necessary. This activity is composed of the following steps.

Steps:



**Figure 5.8** RiPLE-EM CAD: Release Execution Flow

- **Gather all Change Requests that are being release in this version:** Gather all CRs and PRs that were resolved in the version being released and include it in the release notes.
- **Gather all known issues (CRs and PRs):** Gather all known CRs and PRs that were NOT resolved in the version being released and include it in the release notes.
- **Consolidate Release Notes:** The Release Notes is being created and updated since the beginning of the release planning, and now it needs to be completed and consolidated in order to be published along with the release distribution package. This activity is composed of the following steps.

Steps:

- **Include Core Asset Description:** It describes the core asset with enough details.
- **Explicit the dependencies of the component:** Describes the direct, first level dependencies of a component on other components. It should be noted that specifying a remotely released component as a dependency is as easy as specifying a locally released component as a dependency, since the release database is transparently distributed.
- **Inform the source location of the component:** Document the specific internal (repository) and external (release publication) location of the core asset being released.
- **Categorize Component:** Categorize Component according to: Release groups (Variants), Associated License, ...
- **Consolidate Release Notes Generation:** Review release notes and complete it in order to be published along with the release notes.

Before the release publication, the build procedures previously defined has to be followed in order to produce a release build, which will be published next.

- **Publish Release:** Publish release and make it available to internal/external clients/users. The release distribution can be done through a variety of ways, such as publishing in a web site, deploying in a production server (web applications), and DVD hand-out direct to customer or users. Despite of the way it is delivered, all involved

stakeholders should be notified about this release publication. This activity is composed of the following steps.

Steps:

- **Publish the release:** The release distribution package and the release notes should be made available for the customers/clients/users. It can be published through: Email, Website, SCM Systems, etc...
- **Notify all SPL Stakeholders of the release production:** After the release is available, all stakeholders should be notified that the new release is available.
- **[optional] Open PR to propagate this new release to PD:** Open PR to propagate the new release to PD.

After the release publication, there is the possibility of propagating the changes done in this release to other products, by opening a PR through the Change management activities.

## 5.6 Chapter Summary

The focus of RiPLE-EM for CAD activities is to guarantee the proper evolution of core assets, from the moment of their creation, passing through various changes (evolution), until the moment the core asset is released for product development and then it start the maintenance phase by following again the iterative RiPLE-EM (from the next release planning, changing, building and releasing).

As stated before, since RiPLE-EM is a release-oriented process model, each core asset development will finish with its release for the product development team to (re)use it. RiPLE-EM comprises the whole cycle from the release planning of the core asset, until the release execution of the proper core asset.

This cycle (the RiPLE-EM CAD cycle) is composed by the disciplines: *Release Planning*, where all information regarding the release planning of the core asset is gathered and stored; *Change Management*, dealing with both the requisition and processing of changes and propagation requests; *Build Management*, where the build of the core asset is executed, for different purposes; and *Release Execution*, where the core asset release to the Product Development is performed by publishing the core asset along with its release notes (that contains information such as the production plan of the core asset).

Next Chapter presents RiPLE-EM specific workflows, activities and steps for the evolution management of the product development level of software product lines.

*“Production is not the application of tools to materials, but logic to work”*

Peter F. Drucker (American Educator and Writer)



## RiPLE-EM PD

The RiPLE-EM for PD defines workflows, activities, steps, roles and work products to properly control the evolution and changes of the products throughout their life-cycle.

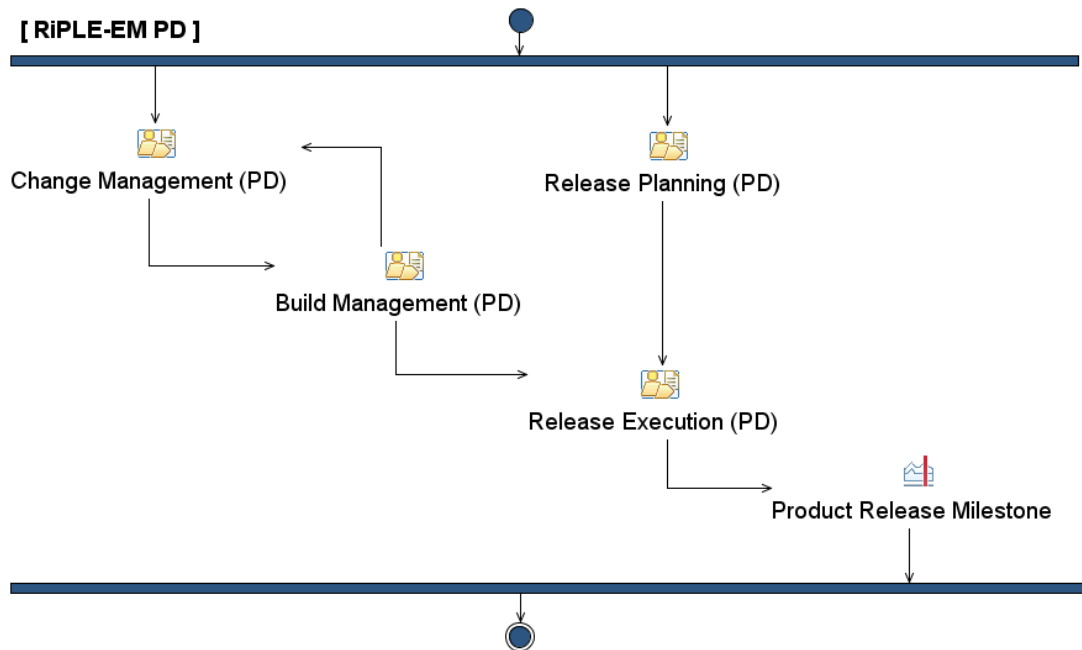
Thus, this Chapter presents in a detailed level, including all flows, activities, and steps of the RiPLE-EM for Product Development. Section 6.1 describes the RiPLE-EM main flow for product development. Section 6.2 discusses the release planning activities, Section 6.3 discusses the change management activities, Section 6.4 presents the build management activities and Section 6.5 details the Release execution activities. The Chapter summary is presented in Section 6.6.

### 6.1 RiPLE-EM for Product Development

The focus of RiPLE-EM for PD activities is to guarantee the proper evolution of products in a product line. RiPLE-EM for PD has the same disciplines as RiPLE-EM for CAD, and follows a similar workflow, but with different activities and different focus.

RiPLE-EM PD, as shown in Figure 6.1 also starts the release planning of the product, where the release schedule and dependencies are defined (e.g. core assets that are still being developed). The development is also supported by change management, enabling the creation of product specific assets, or core asset instances for the product being developed, and build management to support the product construction. In the release execution of a product, activities such as the creation of a release notes, publication and eventual changes propagation are also supported by RiPLE-EM PD.

Each activity in Figure 6.1 is expanded into several activities to achieve specific goals. The activities will be detailed in the following Sections.



**Figure 6.1** RiPLE-EM PD Macro Flow

## 6.2 RiPLE-EM PD - Release Planning

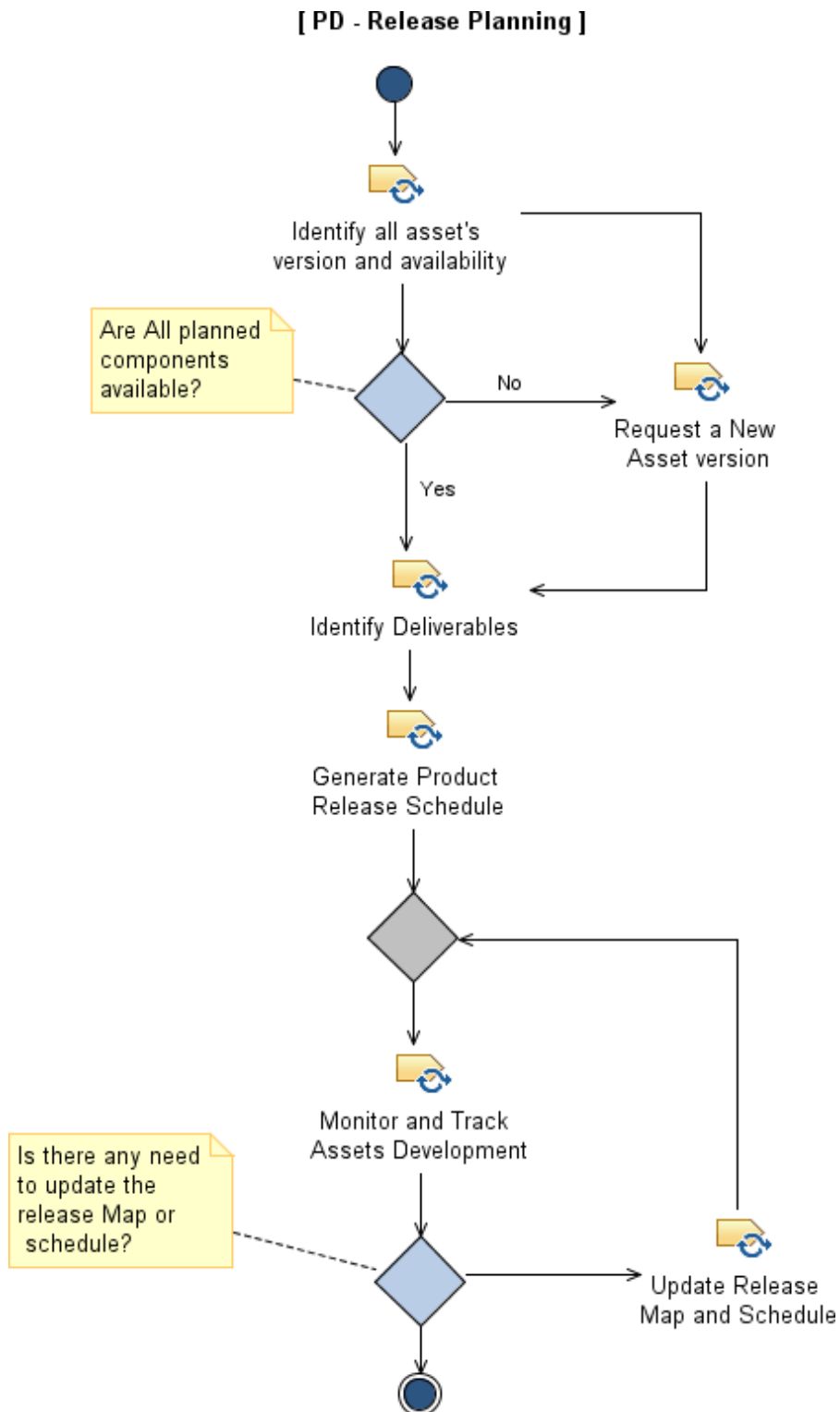
The activity of planning the product release can run in parallel with the proper product development, and the release planning can be refined until the release execution moment.

The activity flow to accomplish the product release planning is shown in Figure 6.2. The activities inside of this flow are described next:

- **Identify all asset's version and availability:** In this activity, all core assets used in the product are identified along with its version and its availability. This information have to be recorded in the product *Release Map*. If the core asset chosen needs a modification to be used in the product, a CR must be opened to change the core asset. This activity is composed of the following steps.

Steps:

- **Identify all core assets that will be used in this product release:** Identify all core assets that will be used in this product release and the version of each core asset.



**Figure 6.2** RiPLE-EM PD: Release Planning Flow



- **Document it in the Product Release Map:** Document it in the product Release Map in order to have this kind of information in the future.
- **Identify Core Assets Availability:** Identify if every core asset planned to be part of this release is available or not at the moment.
- **Request a New Asset version:** If needed, the product team, can request a change in a core asset or request a product specific instance creation of a certain asset. This activity is composed of the following steps.

Steps:

- **This activity extends the “Request Change” activity in section 6.3.**
- **Identify Deliverables:** Identify all items that will be part of the release package/distribution. Knowing the items are going to be inside the release distribution package is vital to the build automation script creation.

It is important that the package creation is automatic, so it essential to know the parts of the release package in advance to automate it. This activity is composed of the following steps.

Steps:

- **Identify Items to be included in release package:** All files that the release package will contain, needs to be identified and documented in the draft release notes. This information is useful to prepare the build script packaging automation. This information can be updated in the draft release notes.
- **Generate Product Release Schedule:** Based on the Product Release Map, the release schedule has to be generated taking the major milestones in consideration. This activity is composed of the following steps.

Steps:

- **Generate Product Release Schedule:** Generate the product release schedule according to the major milestones, taking in consideration all constraints and restrictions.
- **Monitor and Track Assets Development:** If the product depends on an asset that is still being developed, this development needs to be tracked and followed to prevent any surprises at the release preparation moment.

E.g. If the date, set to start the final test round in a core asset release, is missed, the product development should know this and re-plan the product release schedule (the product that has that core asset). This activity is composed of the following steps.

Steps:

- **Verify if the core asset development is complete.** Keep track of the core assets release to identify if the version needed/requested is already released.
- **Verify if the core asset development is following the schedule.** If the core asset is not released until the moment, the ideal is to see if the development is on schedule or if it has any problem or delay.
- **Update Release Map and Schedule:** Every time a problem or a delay is detected in a core asset release (that this product depends on), the product release schedule should be updated. This activity is composed of the following steps.

Steps:

- **Reflect the changes in the product release map:** The product release map has to be updated if any new core asset is going to be used in the product.
- **Reflect the changes in the product release Schedule:** The product release schedule may to be updated if there was any problem with the core asset being developed/changed for the product.

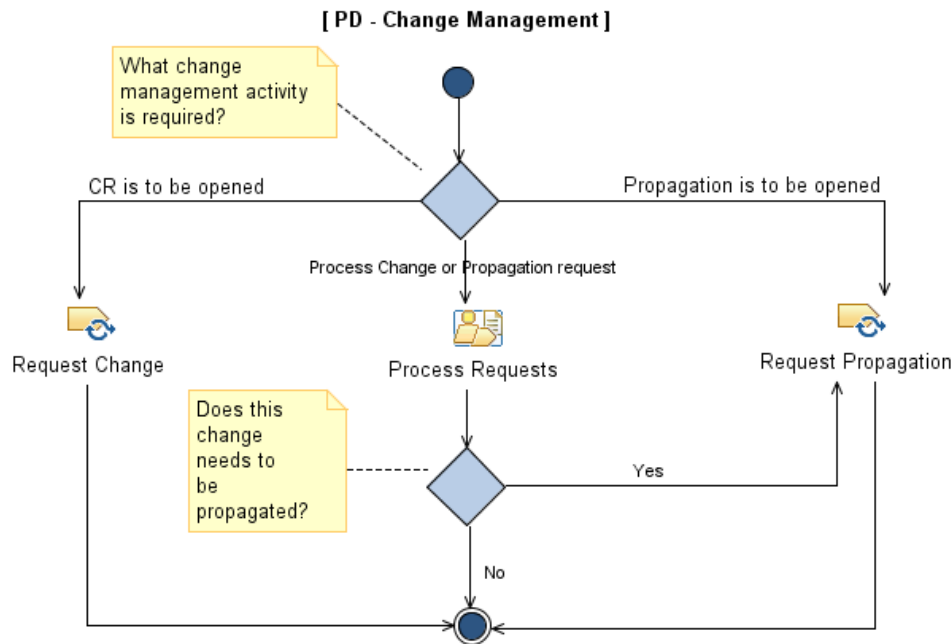
## 6.3 RiPLE-EM PD - Change Management

When the development of the product takes place, in other words, the creation of the product and its evolution (changes), the change management flow starts.

The change management activity is the activity which interfaces with the implementation (realization) of the software product line, where the changes really occur, products are created, core assets instantiated and modified (evolved).

In order to control the product changes, the flow showed on Figure 6.3 is proposed. The activities inside this flow are expanded and described next.

As the RiPLE-EM CAD change management flow, the RiPLE-EM PD change management flow starts with the *change request*, *propagation request*, or *process requests* (To *change request* and *propagation request* details refer to Section 5.3 ).



**Figure 6.3** RiPLE-EM PD: Change Management Flow

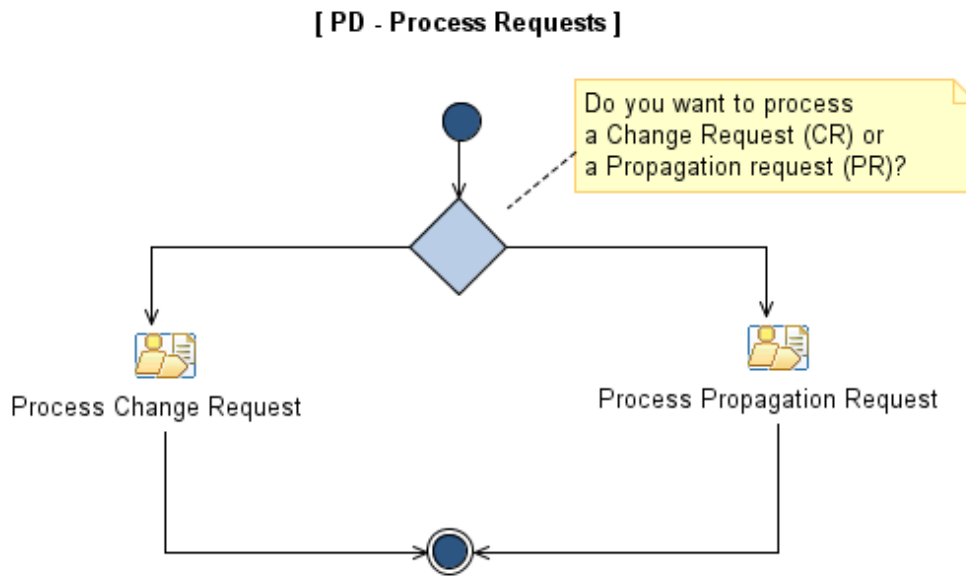
When the *process request* is expanded, there are two possibilities are available, according to picture 6.4.

Expanding the *Process Change Request*, a new flow is started, as shown in Figure 6.5. The activities of this flow are detailed next.

- **Analyze Change Request:** For more information about this activity, refer to 5.3.
- **Delegate Changes:** For more information about this activity, refer to 5.3.
- **Manage Asset Artifacts:** For more information about this activity, refer to 5.3.
- **Create a new Instance:** When changes need to be done at the product level, instances of core assets needs to be created. This instance creation means a new branch (or any other form of instantiation) for the development of specific product code/documentation. This activity is composed of the following steps.

Steps:

- **Identify from which core asset release version the instance will be created.**

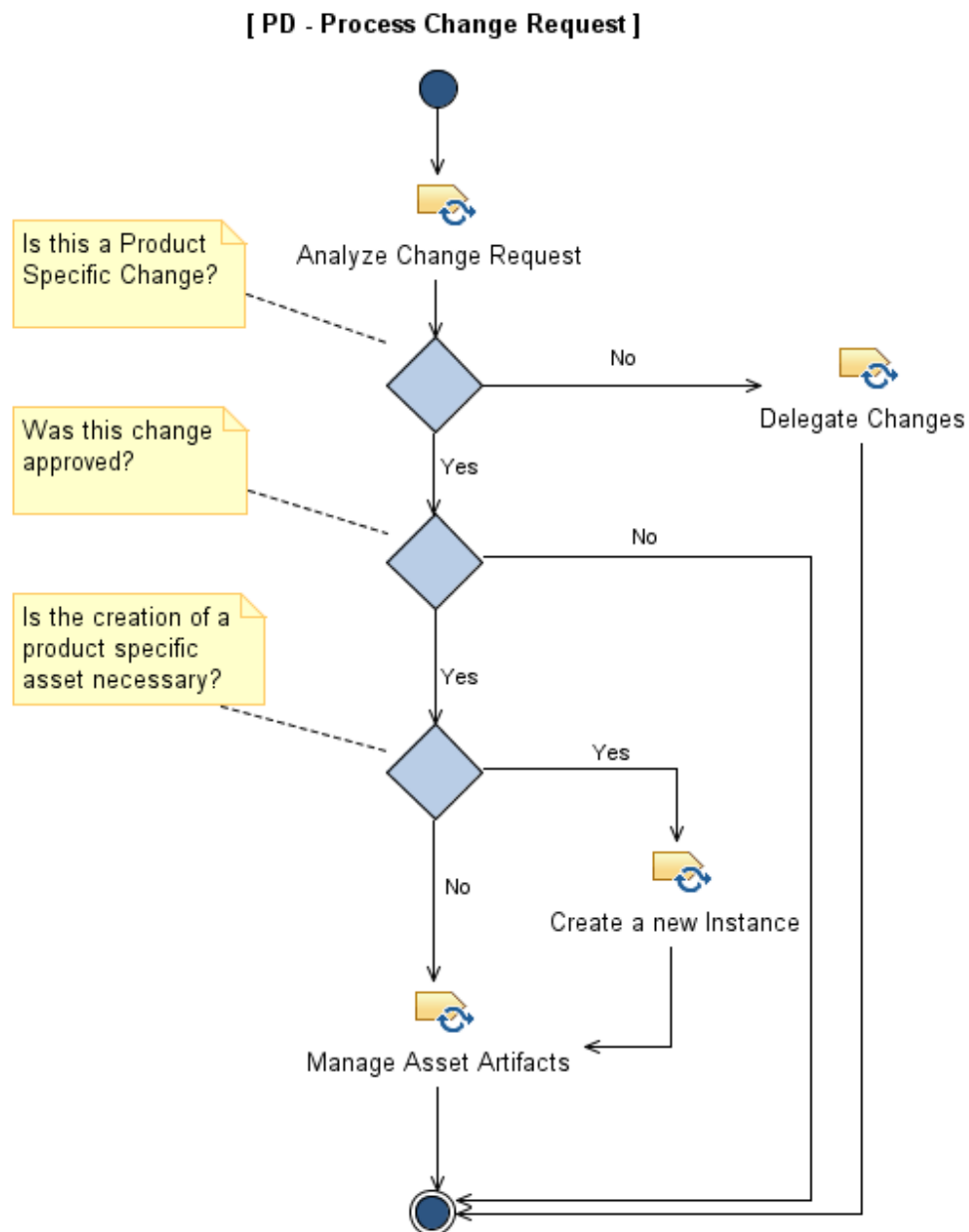


**Figure 6.4** RiPLE-EM PD: Change Management - Process Requests

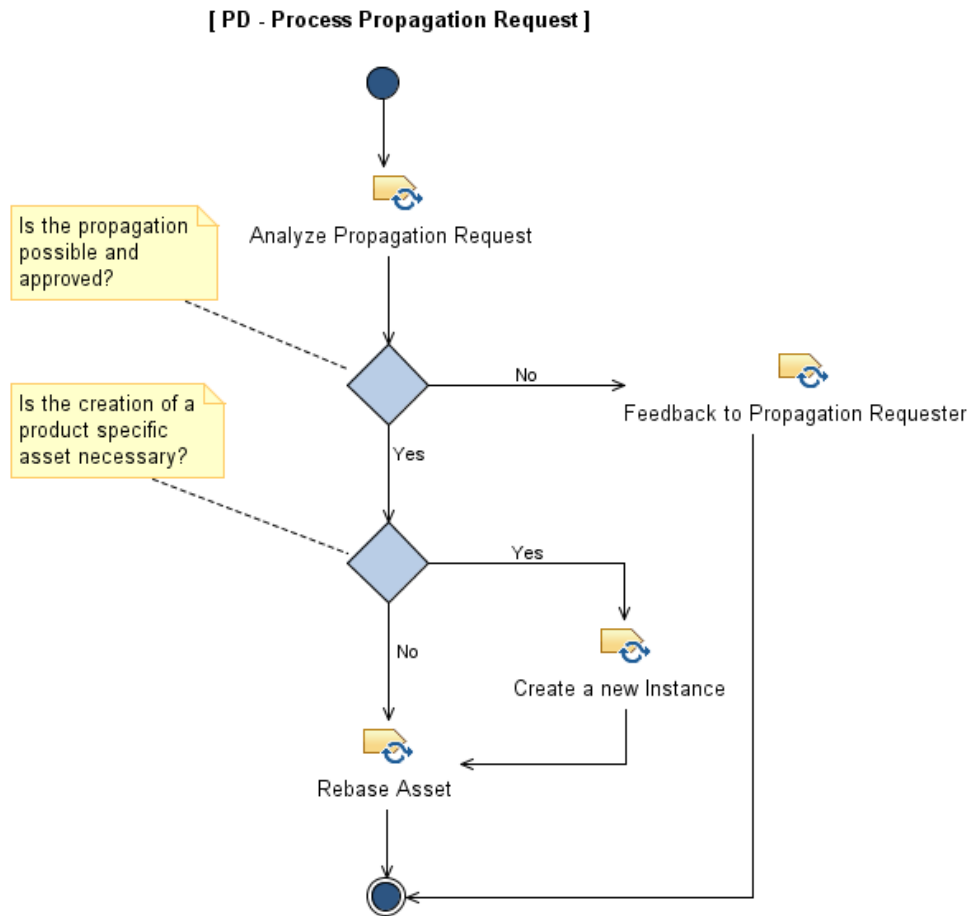
- **Explicit the product that is generating the instance:** In the instantiation, the product that drives the instantiation must be explicitly documented. A possible suggestion is include the product name (or identification) in the branch naming scheme (if branches are used). E.g. The branches could be named like “**inst\_productA**” where “**inst**” stands for **instantiation** and “**productA**” stands for the name of the product instantiating the core asset.

In addition, expanding the *Process Propagation Request* activity, a new flow is started, as shown in Figure 6.6. The activities of this flow are detailed next.

- **Analyze Propagation Request:** For more information about this activity, refer to 5.3.
- **Feedback to Propagation Requester:** For more information about this activity, refer to 5.3.
- **Rebase Asset:** After a core asset release execution, it is common for product teams to receive a Propagation Request to rebase their instances with a more recent (generally latest) release version. Two types of rebasing propagation are possible: *Total replacement* of the core asset by its new release version in cases



**Figure 6.5** RiPLE-EM PD: Change Management - Process Change Request



**Figure 6.6** RiPLE-EM PD: Change Management - Process Propagation Request

where the asset was only (re)used and not instantiated (changed); or *Merge* changes between CAD version and PD instantiated version. This activity is composed of the following steps.

Steps:

- **Propagate changes from CAD to PD:** This activity is the actual propagation of changes and must follow the propagation type defined in the propagation analysis.
  - **[optional] Resolve risen conflicts:** Only performed if the propagation type is Merge. All risen conflicts must be resolved in order to promote changes to a build to be tested.
- **Create a new Instance:** This activity was detailed previously in this Section.

## 6.4 RiPLE-EM PD - Build Management

When the changes proposed are complete, a build of the product should be generated for different purposes (for developer testing, for integration testing or for releasing).

It is important to observe that some existing tools (such as gears (Krueger, 2008) and pure::variants (Beuche, 2008)) in the SPL build management area already substitute (automatically) most of the activities in this flow. Thus, this flow can be replaced by appropriate and automatic tools.

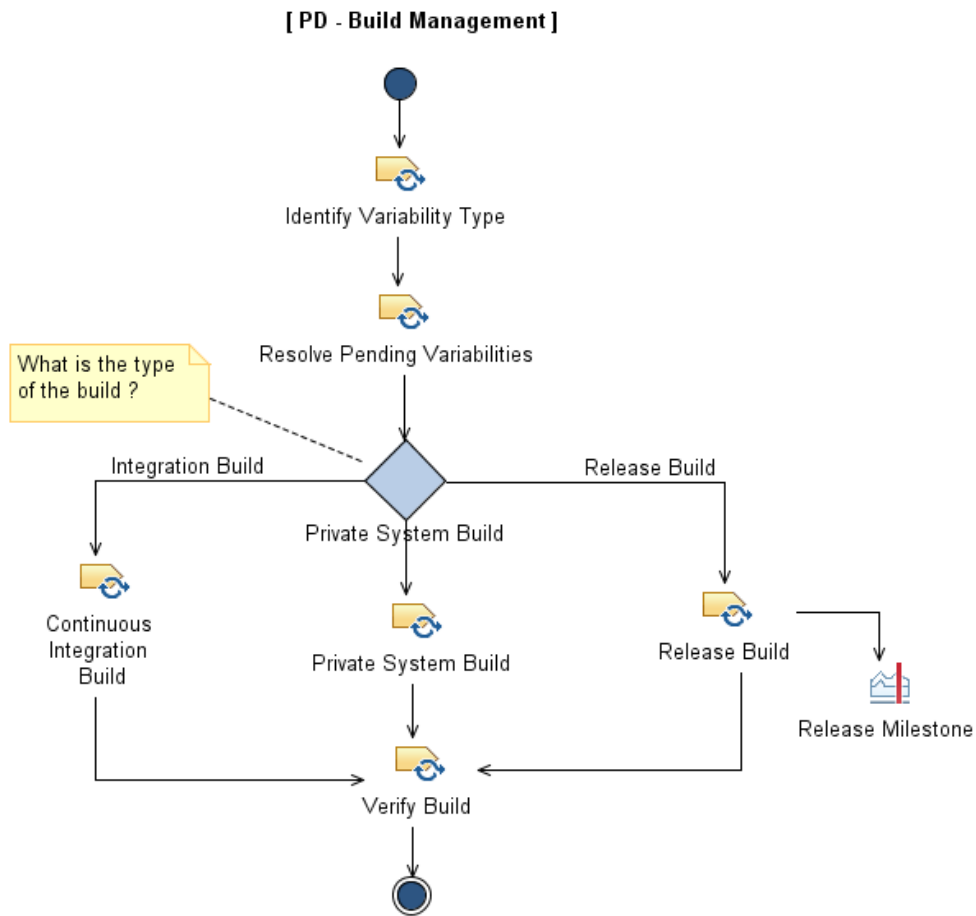
RiPLE-EM activity flow to guide build management activities is illustrated in Figure 6.7. The main and only difference to the RiPLE-EM for CAD build management flow, is the *Identification of the Variability Type* before others activities.

For the build management activities details see Section 5.4, where the activities are described.

In order to proper generate product builds (for testing or release purposes), the variability still left in the core assets have to be resolved, and configured.

The way the variability was implemented, and the way it is going to be resolved are not the focus of this work, thus, the set of activities are only a high level set of activities with the goal of having a *buildable product* after the configuration. Therefore, any variability technique can be used in these high level activities.

- **Identify Variability Type:** The variability provided by the core assets must be resolved in order to generate a version with the right decisions for the product.



**Figure 6.7** RiPLE-EM PD: Build Management Flow



This activity aims at identifying the existing variation points, analyze the possible decisions and pick the right decision for the product being produced, if it was not done already.

Steps:

- **Identify Variability in order to resolve part (or all) of them.**

## 6.5 RiPLE-EM PD - Release Execution

Once the build package is tested and approved for release (to the product development), the RiPLE-EM CAD Release Execution flow takes place.

RiPLE-EM activity flow to guide release execution activities is illustrated in Figure 6.8. All activities in this flow are explained next.

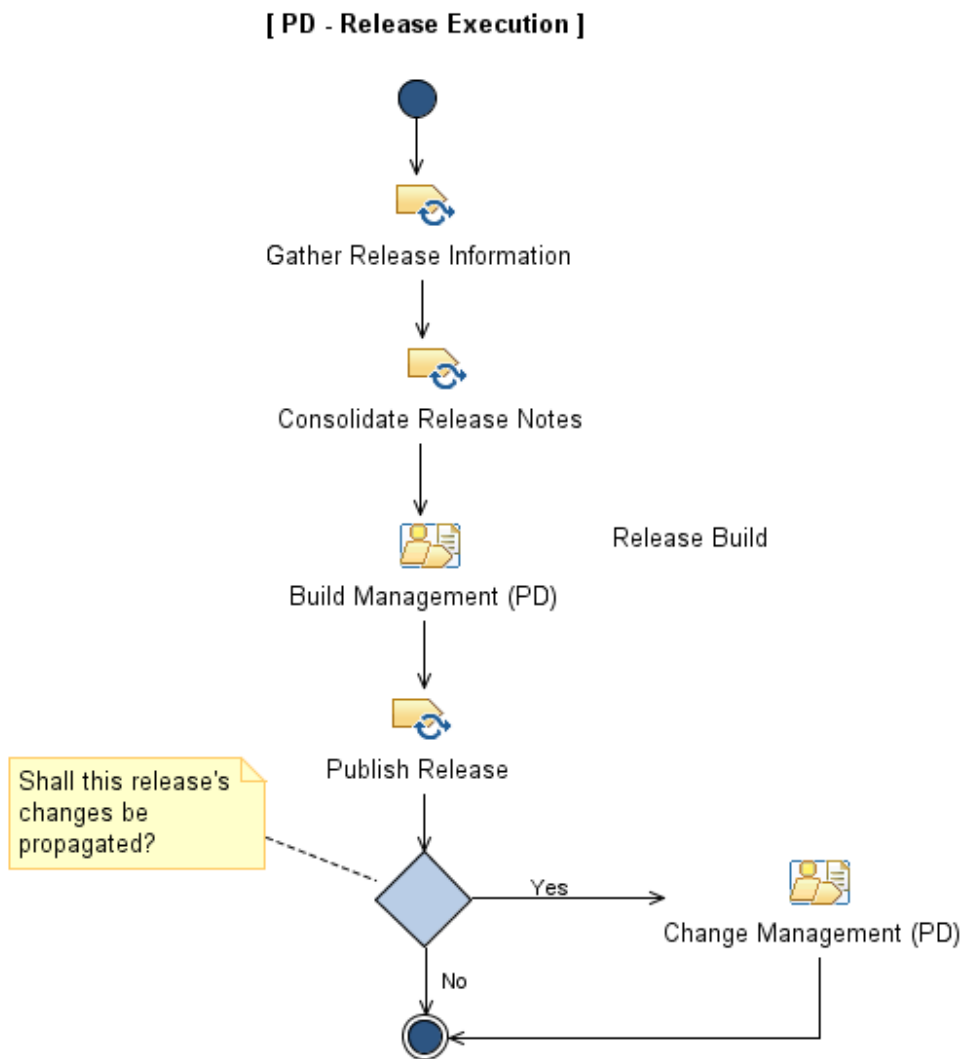
- **Gather Release Information:** Gather Information about the Release being produced. Some information are already inside the draft release notes, but still, some are necessary. This activity is composed of the following steps.

Steps:

- **Gather all Change Requests that are being release in this version:** It should gather all CRs and PRs that were resolved in the version being released and include it in the release notes.
- **Gather all known issues (CRs and PRs):** It should gather all known CRs and PRs that were NOT resolved in the version being released and include it in the release notes.
- **Consolidate Release Notes:** The Release Notes is being created and updated since the beginning of the release planning, and now it needs to be completed and consolidated in order to be published along with the release distribution package. This activity is composed of the following steps.

Steps:

- **Include Product Description:** Give an overview of the product being released.
- **Inform the location of the product:** Document the specific internal (repository) and external (release publication) location of the product being released.



**Figure 6.8** RiPLE-EM PD: Release Execution Flow

- **Consolidate Release Notes Generation:** Review release notes and complete it in order to be published along with the release package.

Before the release publication, the build procedures previously defined has to be followed in order to produce a release build, which will be published next.

- **Publish Release:** Publish release and make it available to internal/external clients/users. The release distribution can be done through a variety of ways, such as publishing in a web site, deploying in a production server (web applications), and DVD hand-out direct to customer or users. Despite of the way it is delivered, all involved stakeholders should be notified about this release publication. This activity is composed of the following steps.

Steps:

- **Publish the release:** The release distribution package and the release notes should be made available for the customers/clients/users. Publish the release through: Email, Website, SCM Systems, etc.
- **Notify all SPL Stakeholders of the release production:** After the release is available, all stakeholders should be notified that the new release is available.
- **[optional] Open PR to propagate this new release to CAD:** It should open the PR to propagate this new release to CAD.

After the release publication, there is the possibility of propagating the changes done in this release to other products, by opening a PR through the change management activities.

## 6.6 Chapter Summary

The focus of RiPLE-EM for PD activities is to guarantee the proper evolution of products, from the moment of their creation, passing through various changes (evolution), until the moment the product is released for a client/market and then it start the maintenance phase by following again the iterative RiPLE-EM (from the next release planning, changing, building and releasing). RiPLE-EM comprises the cycle from the release planning until the release execution of the product.

This cycle (the RiPLE-EM PD cycle) is composed by activities of: *Release Planning*, where all information regarding the release planning of the product is gathered and stored;

---

*Change Management*, dealing with both the requisition and processing of changes and propagation requests; *Build Management*, where the build of the product is executed, for different purposes; and *Release Execution*, where the product release is performed by publishing the product along with its release notes.

Next Chapter presents the experimental study definition, planning, and conduction. This experimental study served as the initial validation of RiPLE-EM.

*“No amount of experimentation can ever prove me right; a single experiment can prove me wrong”*

Albert Einstein (1879-1955)

# 7

## The Experimental Study

In order to experiment the RiPLE-EM process, an experimental study was conducted following the experiment process proposed by (Wohlin *et al.*, 2000).

Wohlin *et al.* defined a process to facilitate experiments conductions, and follows the phases described next:

- **Definition:** The experiment objectives and goals are defined, as well as the purposes and the hypothesis statement.
- **Planning:** In this phase, the hypothesis has to be formally stated, including a null and an alternate hypothesis. All variables and treatments for those variables are defined, and the experiment subject is selected. The threats for the experimentation validity are also explicated in this phase.
- **Operation:** The operation phase consists in the execution of the experiment and validation of the data collected.
- **Analysis and Interpretation:** The objective of this phase is to analyze and interpret, by means of descriptive statistics, or qualitative analysis, the data collected in the operation phase.
- **Presentation and Package:** This phase concerns with presenting and packaging the experiment findings, including guidelines on how to document and publish the experiment results.

All these phases were executed and are described in the following sections. Section 7.1 describes the definition of the experimental study and goals. Section 7.2 details the planning of the experiment, with the experiment hypothesis. Section 7.3 describes the

operation phase, and Section 7.4 presents the analysis of the collected data. Some lessons learned are described in Section 7.5 and the chapter summary is presented in Section 7.6.

## 7.1 The Definition

In the definition phase the foundation of the experiment is determined. The main purpose of this phase is to define the goals of the experiment. For the goals definition, (Wohlin *et al.*, 2000) follow the GQM model (Basili *et al.*, 1994).

A GQM model is a hierarchical structure that starts with a goal (conceptual level), and this goal is refined into several questions (operational level), that usually breaks down the issue into its major components (Basili *et al.*, 1994). Each question is then refined into metrics (quantitative level), that will characterize the goal in a quantifiable way.

Besides using GQM, (Wohlin *et al.*, 2000) defines a goal template for the goal definition. The goal template comprises the definition of the objects of study, purposes, quality focus, perspective and context, which are detailed in the next sub-sections.

### 7.1.1 Goal

The goal of this experiment is to analyze the use of RiPLE-EM regarding different viewpoints and topics further described in this Section.

**Object of study.** The object of study in this experiment is the RiPLE-EM process, and its activities, steps, artifacts and roles.

**Purpose.** The purpose is to evaluate the RiPLE-EM use, to verify its applicability in a SPL project. The purpose is also, to collect metrics in order to improve the process understandability, completeness, applicability and effectiveness, and minimize the risks of applying it in a real and critical scenario.

**Perspective.** In this experiment, the perspective is from the point of view of the researcher. Another perspective is from the point of view of a build and release engineer (some times these activities are performed by the configuration management engineer).

**Quality Focus.** The main effect studied in the experiment is the understandability, completeness, applicability and effectiveness of the evolution of products and assets inside a SPL, maintaining its integrity, and assuring the right communication between core assets development and product development.

### 7.1.2 Questions

In order to evaluate RiPLE-EM process, to achieve the goal previously defined, some qualitative and quantitative questions were defined and described next.

#### General

**Q1.** How much effort does it take to apply the process?

#### Usability and Understandability

**Q2.** Do the subjects have difficulties to understand/apply the process?

**Q3.** Are the subjects satisfied in using the process?

#### Completeness

**Q4.** Is there any missing activity, roles or artifact?

#### Effectiveness

**Q5.** How many propagation requests were uncompleted?

### 7.1.3 Metrics

Once the questions have been developed, the next step is to associate the questions with appropriate metrics. The metrics are quantitative ways to answer the questions.

#### M1. Effort to Apply the Process (EAP)

This metric relates to question **Q1**. This metric measures the amount of time spent in order to understand and follow the RiPLE-EM process and produce the artifacts proposed by the process.

$$EAP = \frac{TotalTimeSpentApplyingRiPLE-EM}{TotalTimeSpentInTheProject}$$

#### M2. Process Understanding and Application Difficulties (PUAD)

This metric relates to question **Q2**. In order to identify possible misunderstandings in the process usage, it is necessary to identify and analyze the difficulties found by users when learning the process.

**PUAD** = Number of subjects with difficulties raised during the process learn and application (and the difficulty distribution).

**M3. Subjects Satisfaction (SS)**

This metric relates to question **Q3**. Satisfaction is the user's response to interaction with the process, and it includes attitudes towards the use of the process. Thus, this metric is proposed to evaluate the subjects satisfaction in process use.

**SS** = subjects' satisfaction distribution according to a defined scale: very satisfied, satisfied, impartial, unsatisfied and very unsatisfied.

According to [Vegas and Basili \(2005\)](#), the subject's satisfaction should not be considered for establishing a hypothesis, because it can be refuted by means of statistical evidence. Thus, this aspect will be assessed informally, examining the opinions of each subject.

**M4. Activities, Roles and Artifacts Missing (ARAM)**

This metric relates to question **Q4**. This metric aims at identifying the activities, roles and artifacts identified as missing in RiPLE-EM by the subjects. With this metric, we want to identify every activity, role and artifact considered absent from RiPLE-EM in order to include them, depending on the analysis.

**ARAM** = Number missing activity/role/artifact raised during the process application (and their distribution).

**M5. Uncompleted Propagation Requests (UPR)**

This metric relates to question **Q5**. The idea of this metric is to analyze the propagation requests number and their status, to identify the percentage of unsucceeded propagation requests.

**UPR** = The number of uncompleted propagation requests / Number of propagation requests (and their distribution according to the propagation request type).

**Definition Summary**

The experiment definition can be summarized, using the template proposed by [Wohlin et al. \(2000\)](#) as:

Analyze the *RiPLE-EM process*

for the purpose of *evaluation*

with respect to *understandability, usability, completeness, applicability and effectiveness*

from the point of view of *SPL researchers and practitioners*

in the context of *a software product line project*.

---



## 7.2 The Planning

As in all types of engineering activities, the experiment must be planned beforehand. The planning phase of the experimental study, which started after the definition phase, cover the steps described in the next sub sections.

### 7.2.1 Context Selection

The context of an experiment can be characterized according to four dimensions: (i) Off-line vs. on-line; (ii) Student vs. professional; (iii) Toy vs. real problems; (iv) Specific vs. general. RiPLE-EM experiment, fits the following categories: **Off-line, Student, Toy and Specific** Wohlin *et al.* (2000).

Still according to Wohlin *et al.* (2000), the context can also be characterized in terms of the number of subjects and objects involved in the experiment. According to this characterization, RiPLE-EM experiment is a **single object study**, which means that the experiment will be conducted on a single subject and a single object study.

This experiment will be performed in the context of a post-graduation course at the UFPE (Federal University of Pernambuco)<sup>1</sup>, Brazil. In this course, a software factory will be created, and will use RiPLE-EM as the process for SPL evolution management. Along with RiPLE-EM, other processes from other disciplines will be also used, such as, a requirements and design.

Inside this post-graduation course, 3 months will be dedicated to the SPL project development. All participants are M.Sc. and Ph.D. students of Computer science, and the subjects of this experiment are further discussed in section 7.2.4.

### 7.2.2 Hypothesis

Two hypotheses have to be formulated: Null Hypothesis, which the experimenter wants to reject with as high significance as possible; and Alternative Hypothesis, which is in favor of which the null hypothesis is rejected.

**Null Hypothesis.** In this study, the null hypothesis determine that the use of the RiPLE-EM in SPL projects does not produce benefits that justify its use, presenting a poor understandability, effectiveness, completeness and applicability. It is important to emphasize that these metrics were never used before, and thus there is no well-known

---

<sup>1</sup>Informatics Center of Federal University of Pernambuco - <http://www.cin.ufpe.br>

value for it and an arbitrary value was chosen, based on practical experience and common sense. The following null hypothesis are defined:

$$H0_1: \mu_{EAP} \geq 20\%$$

$$H0_2: \mu_{PUAD} \geq 40\%$$

$H0_3: \mu_{ARAA} < 3$  (Having in mind that RiPLE-EM has 34 different tasks, we came to the value of 10% (3 activities) as being a reasonable number.)

$H0_4: \mu_{UPR} > 20\%$  (The number of uncompleted propagation requests will be only influenced by process' issues. If the propagation request was not completed due to the analysis of the propagation, it will not be counted.)

**Alternate Hypothesis.** This is the hypothesis in favor of which the null hypothesis is rejected. In this study, the alternative hypothesis determines that the use of the process produces benefits that justify its use. Thus, the following hypothesis can be defined:

$$H1_1: \mu_{EAP} < 20\%$$

$$H1_2: \mu_{PUAD} < 40\%$$

$$H1_3: \mu_{ARAA} > 3$$

$$H1_4: \mu_{UPR} < 20\%$$

### 7.2.3 Variables and Treatments

In the variables selection, we choose the independent and dependent variables. All variables in a context that are manipulated and controlled are called independent variables. In this study, the independent variables are the experience of the subjects, and the proposed process.

The dependent variables are the variables that we want to study to see the effect of the changes in the independent variables. In this study, the dependent variables are the understandability, usability, effectiveness, completeness and applicability of the proposed process, in terms of evolution management (change, build and release management).

### 7.2.4 Subjects

The subjects of the study are M.Sc. students and Ph.D. students. They will be requested to act as the roles defined in the process according to their experience and interest.

They were selected by convenience sampling [Wohlin et al. \(2000\)](#) representing a non-random subset of software engineering students universe, characterizing a *quasi-experiment*.

### 7.2.5 Experiment Design

A design of an experiment describes how the tests are organized and run. The general design principles are randomization, blocking and balancing.

**Blocking and Balancing.** Some of the design principles such as blocking and balancing, are not applicable to this study, since the study will evaluate one factor with only one treatment, thus, the subjects will not be divided into different blocks.

**Randomization.** Randomization is the only applicable design principles, cited by Wohlin *et al.* (2000), and was applied in the subjects selection.

### 7.2.6 Instrumentation

In this study, the RiPLE-EM documentation will be available for the subjects to execute the proposed activities and steps, as well as checklists. In addition to that, all subjects will be trained to use RiPLE-EM by a process expert. The subject's training will be conducted in an university classroom, and will be divided into two steps: (i) Concepts related to software reuse, software product lines, variability and domain engineering; (ii) and RiPLE-EM flows, activities, artifacts and roles.

The instrumentation of this experiment also includes questionnaires that will be filled by the subjects and will serve as data for future analysis. The majority of the data collected in these questionnaires will serve as basis for the hypothesis assessment. All questionnaires used in this experiment are available in Appendix A.

All instruments available in this experiment are listed as follows.

- *RiPLE-EM Documentation:* A complete on-line description of RiPLE-EM process was provided, with all support material such as templates and checklists.
- *Training:* The training about reuse in general, SPL and RiPLE-EM process, and its activities.
- *Background form:* This form is intended to collect the subjects background information.
- *Process evaluation questionnaire:* In order to evaluate RiPLE-EM, the subjects will answer this questionnaire.
- *Time sheet:* All time spent within the project will be recorded in this sheet.

- *Support tools:* To support RiPLE-EM use, some tools are essential. In this case, the tools used were Subversion (SVN) <sup>1</sup> for version control, Trac <sup>2</sup> for changes, propagation requests and bug tracking, and the CL [Anastasopoulos \*et al.\* \(2009\)](#) for the encapsulation of SPL activities regarding version control.

### 7.2.7 Validity Evaluation

Following ([Wohlin \*et al.\*, 2000](#)) process, one important part of the experiment planning is the identification of the validity threats. Adequate validity refers to that the results should be valid for the population of interest, first of all, the population from which the sample is drawn, and secondly, if possible, generalize the results to a broader population.

According to ([Wohlin \*et al.\*, 2000](#)), there are different classification schemes for the different validity threats of an experiment:

- **Conclusion validity:** This validity is concerned with the relationship between the treatment and the outcome.
- **Internal validity:** This validity is concerned with the any relationship observed between the treatment and the outcome, to make sure that every relationship is merely causal.
- **Construct validity:** This validity is concerned with the relationship between theory and observation.
- **External validity:** This validity is concerned with generalization. If there is a causal relationship between the construct of the cause, and the effect, can the results of the study be generalized outside the scope of the study?

The validity threats identified in this experiment are listed next, classified by the schemes presented before:

#### **Conclusion validity**

- **Fishing:** Searching or fishing for specific results is a threat since the analyses are no long independent, and the researchers may influence the results by looking for a specific outcome. Since results stating that the treatment used is better than using no treatment for the specific factor, this threat was identified and the formal experiment definition is one of actions to minimize this threat.

---

<sup>1</sup>Subversion project home - <http://subversion.tigris.org/>

<sup>2</sup>Trac project home - [trac.edgewall.org/](http://trac.edgewall.org/)

---

- **Reliability of measures:** The validity of the experiment is highly dependent on the reliability of the metrics used. Objective metrics are more reliable than subjective metrics, for the reason that objective metrics are more reproducible than subjective metrics (eg. if we measure a certain phenomenon twice, it should provide the same outcome). In this experiment most of the question and metrics are not objective and quantitative, but we tried to minimize this threat by categorizing the subjective answers into levels.
- **Reliability of treatment implementation:** The implementation of the treatment can be a threat when the implementation is not similar between the different persons applying the treatment. Thus, the treatment implementation in this experiment was as standardized as much, having the same kind of training for the treatment implementation for each subject.
- **Heterogeneity of subjects:** Group with high heterogeneity present a risk to conclusion validity, since the outcome variation due to individual differences is larger than due to the treatment. This study is composed by M.Sc. students and Ph.D. students that normally have similar knowledge and background, which aid to reduce the heterogeneity. Subjects without experience also can affect this validity, since it is harder for them to apply the treatment. To mitigate the lack of experience, we will provide training in software product lines and evolution management.

### Internal validity

- **Maturation:** This threat concerns with the team commitment and motivation. Since the experiment will last approximately 4 months, there is always the risk of lack of motivation and decreasing commitment of the team.
- **Mortality:** If too many people leave the study, this can be threat to the validity. We have not seen any systematic trend in people leaving the course. To our knowledge, the people who left the course did this independently of the used process. Therefore, this threat is not considered important for the experiment.
- **Instrumentation:** The artifacts that are going to be used can affect negatively the experiment, thus, the artifacts were carefully designed, trying to turn subjective questions into measurable questions (with levels to choose instead of completely subjective questions).

- **Selection:** There were no volunteers in participating in the experiment. Thus, the selected group is more representative for the whole population (since volunteers are generally more motivated and may influence the results).
- **Time Constraints:** Given the fact that the project will start from the initial phases of a software product line, and also the fact that the time available for the project (as detailed in section 7.1.1) was not long, it may occur that not all the activities of RiPLE-EM will be executed. Since only one product will be demanded from the subjects, RiPLE-EM for PD activities can not be taken in consideration by the subjects.

### Construct validity

- **Data Bias:** The experiment is part of a course, where the students are graded. This implies that the students may affect their data, as they believe that it will give them a better grade. It was, however, in the beginning of the course emphasized that the grade depend the subject's participation and effort, timely and proper delivery, analyzing also if it is in compliance with the process.
- **Mono-Operation Bias:** Since the experiment includes a single treatment, it may under-represent the construct, and thus not give the full picture of the theory.
- **Interaction of different treatments:** The subjects were exposed to multiple experiments at the same time, with different and decoupled variables and treatments. The variables are independent, this way the each treatment to each value will not interfere on each other.
- **Evaluation Apprehension:** The subjects of the experiment are post-graduate students, and the experiment context is a post-graduation course. This way, because of the grades, some of them may find themselves a little afraid, or try to look better, and that may confound the outcome. To minimize this risk, the students will be told that the evaluation forms completion will not interfere on their grades.
- **Experimenter Expectancies:** Surely the experimenter expectancies may bias the results, and for that reason, all formal definition and planning of the experiment is being carefully designed beforehand, and reviewed by other students (performing other experiments).

### External validity

---

- *Generalization of subjects and scope.* The experiment will be conducted on a defined time according to the schedule of the course, which could affect the experiment results. The SPL scope will be defined according to this schedule to guarantee the complete execution of the project. Thus, this scenario could have a toy size that will limit the generalization. However, negative results in this scope is a strong evidence that in a bigger scope would fail too. The same applies to the subjects (M.Sc. and Ph.D. students) who normally have similar knowledge in SPL.

## 7.3 The Operation

The operation phase consists of three steps: *preparation, execution and validation*.

### 7.3.1 Preparation

The subjects were seven M.Sc. students and two Ph.D. students. They corresponded to all students taking the reuse course in the postgraduate curriculum. The students were informed that we would like to investigate the outcome of the process application. However, they were not conscious of what aspects we intended to study, i.e. they not aware of the hypotheses stated. Before the experiment was executed, all experiment instruments should be prepared and ready. Thus, all instrumentation defined in Section 7.2.6 were provided.

### 7.3.2 Execution

The experiment was conducted from August/08 to December/08, according to the definition and planning documented, and data was collected. Initially, the subjects were trained in several aspects of SPL and in the applied processes (from August to October 2008), and after, they performed the SPL project from October to December in 2008.

Most of the students had participated in industrial projects. However, the subjects had low or none industrial experience in reuse activities, such as component development and SPL engineering. On the other hand, seven participants are members of the RiSE group, and their research area involve these aspects, which give them theoretical know-how.

Regarding evolution management (and the disciplines in RiPLE-EM), the majority had low experience in both academic and industrial/commercial projects. Table 7.1 shows a summary of subjects' profile.

## 7.4. ANALYSIS AND INTERPRETATION

Subject ID	Years since graduation	Participation in industrial projects	Experience in Evolution Management	Management Knowledge	Experience in Software Reuse
1	3	1 - Medium Complexity 4 - High Complexity	None	None	None
2	5	5 - Low Complexity 2 - Medium Complexity	Low - Academic Medium - Commercial	Version Control (SVN)	Low - Academic Low - Commercial
3	1,7	3 - Low Complexity 1 - Medium Complexity	Low - Academic Low - Commercial	Version Control (SVN), Change Control (Trac, Mantis), Release Management	Medium - Academic Low - Commercial
4	6	1 - Low Complexity 3 - Medium Complexity 3 - High Complexity	Medium - Academic Low - Commercial	Version Control (SVN, CVS), Change Control (Trac), Build Mangement (Ant, Maven).	Medium - Academic Low - Commercial
5	0	-	Low - Academic	None	Medium - Academic
6	2	-	Low - Academic	Change Control (Trac)	Low - Academic
7	2	1 - High Complexity	Low - Academic Low - Commercial	None	Medium - Academic Low - Commercial
8	0,9	1 - Low Complexity 1 - Medium Complexity	Low - Academic Low - Commercial	Version Control (SVN)	Low - Academic Low - Commercial
9	4	1 - Low Complexity	Low - Academic Low - Commercial	None	Low - Academic Low - Commercial

**Table 7.1** Subject's Profile in the Experimental Study

### 7.3.3 Data Validation

Data was collected from 9 students. However, data from 4 students (ID 2,4,6,9 - see Table 7.1) were removed, because they did not participate of the whole study and/or they did not answer the questionnaire related to satisfaction and difficulties in the use of the process. Thus, this may have affected the data validation. Therefore, we left 5 students for statistical analysis and interpretation of the results.

## 7.4 The Data Analysis and Interpretation

After collecting experimental data in the operation phase, we are able to draw conclusions based on this data. The analysis and interpretation of the data are presented as follows.

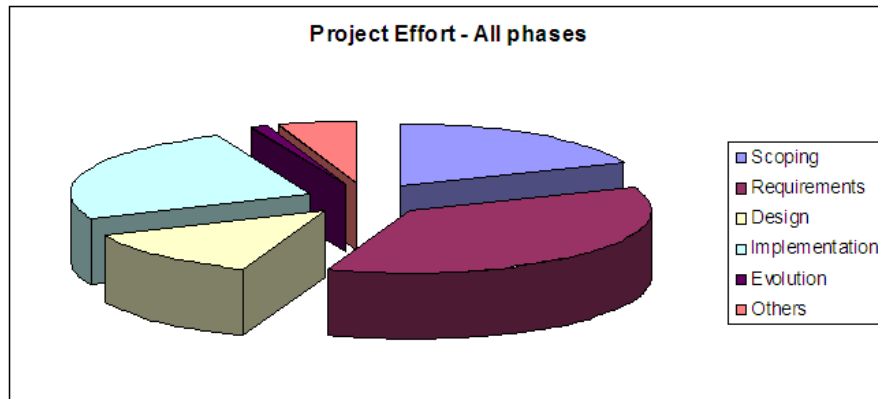
### 7.4.1 Effort to Apply the Process

To develop this project, approximately 590 hours were spent including all hours spent in every phase of the project, as we can see in Table 7.2. In the evolution management activities we may notice that only 7 hours were spent, representing 1,24% of the whole project. With this number, the null hypothesis ( $H0_1: \mu_{CAP} \geq 20\%$ ) was **rejected**.

Even though this hypothesis was rejected, it seems that the effort was too small, indicating that the subjects did not executed all applicable tasks in the project. The reason for this small effort may lies on the fact that the project only had one product as output



Effort	Hours	%
Scoping	111,52	18,93
Requirements	217,01	36,73
Design	74,30	12,61
Implementation	149,30	25,3
Evolution	7,20	1,24
Others	30,40	5,19
Total	589,73	100

**Table 7.2** Effort to use the process**Figure 7.1** Effort Pie Chart

for the product line created, because of time constraints for the project. This validity risk is mapped in section 7.2.7 (*Time Constraints*).

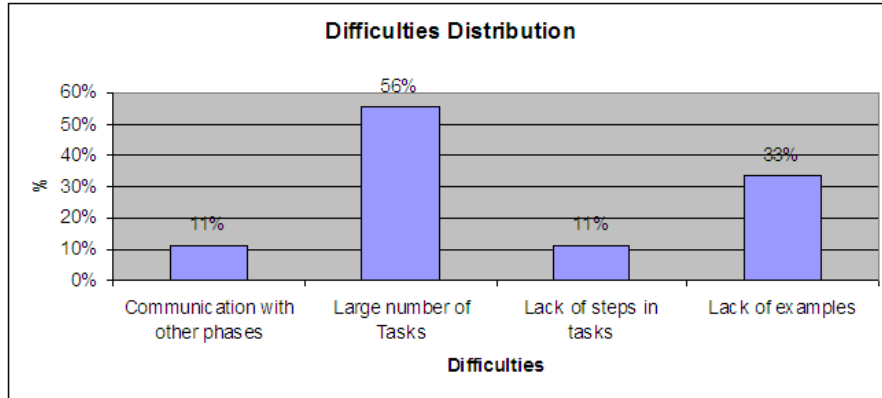
As we can see in the graph represented by Figure 7.1, scoping and requirements were the phases where more hours were spent, followed by implementation and design. Because of this reason, the actual development, coding, building and releasing was postponed until the end of the project.

#### 7.4.2 Difficulties to understand/apply the process

Analyzing the subject's answer in the use of RiPLE-EM, it was identified that all subjects had any kind of difficulty to understand the process. Because of the understanding problem, all of them had also problems to apply the process. The difficulties are summarized in Table 7.3.

Four subjects (ID 1,3,5,7,8) claimed that the main problem to the understandability of the process was the number of activities level and tasks, that were too many and very granular. Another subject (ID 8) stated that one of the understandability issues was the lack of steps in some tasks, making it harder to understand. The lack of examples to the

Difficulty	Number of Subjects
Communication with other phases	1
Large number of Tasks	5
Lack of steps in tasks	1
Lack of examples	3

**Table 7.3** Effort to use the process**Figure 7.2** Difficulties Distribution Chart

process use was another difficulty the subjects (IDs 1,3,5) reported.

The communication of the RiPLE-EM process with other phases of the project, such as requirements, design and implementation was an understandability issue raised by one of the subjects (ID 7).

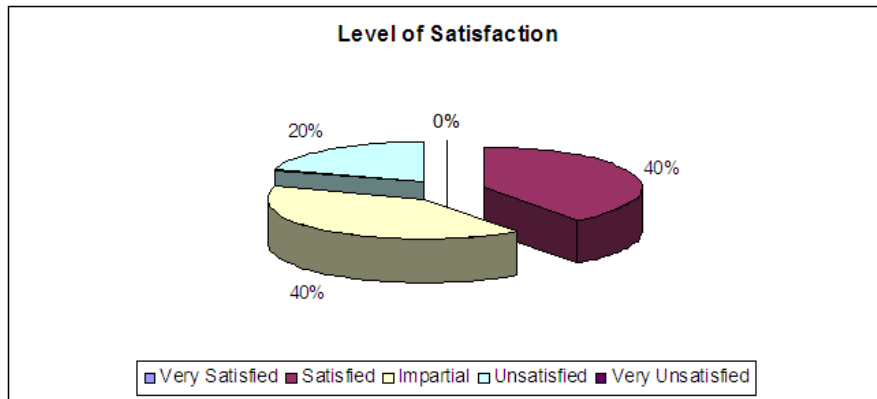
Nevertheless, the next time that the experiment is performed this value can be refined based on this experience, resulting in a more calibrated metric.

Figure 7.2 shows the histogram with the distribution density of the found difficulties.

The null hypothesis related to the percentage of subjects with any kind of difficulty in the process defines a percentage of more than 40% ( $H0_2: \mu_{PUAD} > 40\%$ ). Since we had 100% of the subjects with at least one difficulty, this null hypothesis of **not rejected**. However, in the same way as the previous hypothesis, this value for the null hypothesis was defined without any previous data.

### 7.4.3 Activities, Roles and Artifacts Missing

The idea with this question was collect more information about the RiPLE-EM missing activities. In this direction, the subjects were asked if there was any missing activity (according to their SPL and evolution background).



**Figure 7.3** Subjects satisfaction levels

Analyzing the data we could notice that **none** of the subjects identified any missing activities. Since we had 0 (zero) activities identified as missing, the ( $H0_3: \mu_{ARAM} < 3$ ) null hypothesis is **rejected**.

#### 7.4.4 Uncompleted Propagation Requests

One important aspect of RiPLE-EM is the concept of propagation requests, which is the communication point between core asset and product development, in terms of replicating changes.

This specific metric tries to identify how many of the propagation requests raised were unsuccessful, because of problems issues, and try to understand the problem in order to improve the process and minimize this kind of problems.

No propagation requests were raised in this project, and that **rejects** the null hypothesis ( $H0_4: \mu_{UPR} > 20\%$ ) since  $\mu_{UPR}=0$  ( $UPR = \frac{0}{0}$ ).

Unfortunately, due to the context of the project (where only 5 components and 1 product were created, and there were no clear distinction between core asset and product development) no propagation requests were raised, reducing the outcome reliability.

#### 7.4.5 Subjects Satisfaction

Regarding the subjects satisfaction, 40% (ID 1,7) of the subjects were satisfied, 40% (ID 3,5) were impartial, and 20% (ID 8) were impartial.

The Figure 7.3 presents the frequency of the subjects satisfaction according to the defined scale.

This aspect was assessed informally, examining the opinions of each subject. Thus, it was not considered in the hypotheses.

## 7.5 Lessons Learned

After concluding the experimental study, some aspects should be considered in order to repeat the experiment, since there were limitations in this first execution.

**Project Context.** The project context was the main issue regarding the RiPLE-EM experiment execution. One of the problems was that other process experimentations (a domain requirements engineering process and a domain design process) were running in parallel to RiPLE-EM experimentation in the same project. Thus, there was too much focus on the core asset development phase (also known as domain phase).

Approximatively 69% of the project time, according to Table 7.2, was spent on the SPL scoping, domain requirements engineering and domain design, leaving not much time to product development (where assets are instantiated, modified, and the changes can be propagated). If there was more time to implement products, and evolve the core assets developed, the use of RiPLE-EM would be more applicable, therefore, the experimentation outcomes would be more precise and reliable.

**Training.** The RiPLE-EM presentation was satisfactory, but for the subjects, it seems to be too much information at once. Therefore, it is interesting to have some support materials for the experiment, with examples on how to use the process in different scenarios and activities.

**Pilot Project.** In order to avoid contexts where the project would not be applicable, a pilot project should be executed in advance. The problem is that running pilot projects to cover the activities of RiPLE-EM is not a trivial task. It would consume a lot of time because to completely run RiPLE-EM, it is needed the whole product line infrastructure created (domain requirements being defined, domain design being created, core assets being developed, products being instantiated and etc), specially if the number of participants of this pilot project is low.

**Motivation.** As the project was not short (as 1 day experiments and so), it was difficult to keep the subjects' motivation during all the execution. Thus, this aspect should be analyzed in order to try to control it. A possible solution can be to define some checkpoints during the project.

	Null Hyposthesis	Result	Rejection
H0 <sub>1</sub>	$\mu_{EAP} \geq 20\%$	$\mu_{EAP} = 1,24\%$	Yes
H0 <sub>2</sub>	$\mu_{PUAD} \geq 40\%$	$\mu_{PUAD} = 100\%$	No
H0 <sub>3</sub>	$\mu_{ARAM} > 3$	$\mu_{ARAM} = 0$	Yes
H0 <sub>4</sub>	$\mu_{UPR} > 20\%$	$\mu_{UPR} = 0\%$	Yes

**Table 7.4** Hypothesis Rejection Summary

## 7.6 Chapter Summary

This Chapter presented the definition, planning, operation, analysis and interpretation of the experimental study that evaluated the viability of the RiPLE-EM process. The experiment, summarized in Table 7.4, analyzed, the process understandability, effectiveness and completeness in the context of a software product line project, following RiPLE-EM as the process to manage changes and evolution of all assets and products in the software product line.

Even with the reduced number of subjects (5), and a not very appropriate context, we could identify some directions for improvements, specially regarding understandability, based on the subjects difficulties. However, two aspects should be considered: the study's repetition in different contexts and studies based on observation in order to identify other problems and points for improvements.

The next chapter will present the conclusions of this work, its main contribution and directions for future works.

*“Be not therefore anxious for the morrow: for the morrow will be anxious for itself. Sufficient unto the day is the evil thereof.”*

Matthew 6:34

# 8

## Conclusions

In the late 1960s, the software engineering area started, comprising different engineering disciplines in order to turn software development less *handcrafted*. Among the disciplines, both software reuse and software mass customization were present. In the software reuse area, different approaches have been proposed in order to achieve the well known goals in the software world: high productivity, high quality and low cost [Bayer et al. \(1999\)](#); [Almeida \(2007\)](#); [Weiss and Lai \(1999\)](#).

As mentioned in Chapter 2, software product lines can produce impressive results ([Weiss et al., 2006](#)), however, it is a complex task to manage evolution in the context of software product line, given different complicating factors, such as the fact that a core asset is shared among products, and every change to this asset can have effects on several products ([McGregor, 2003](#)), combined with the fact that in the SPL we have to deal with evolution in time (versions) and space (variability) ([Krueger, 2002](#)).

On the other hand, the efforts towards processes to guide and facilitate systematic evolution management are not strong enough. As stated by the systematic review, detailed in Chapter 3, there is no existing SPL evolution management process definition.

Thus, in order to address the existing problems and challenges in terms of evolution management, this dissertation presented RiPLE-EM - RiSE Product Line Engineering Evolution Management. RiPLE-EM manages evolution in terms of change, build and release management, providing systematic activity flows, steps and roles in order to maximize the SPL benefits.

This Chapter remainder is organized as follows. The research contributions are highlighted in Section 8.1. The work related to RiPLE-EM are described in Section 8.2 and the future work concerning RiPLE-EM and evolution management in SPL are listed in Section 8.3. Academic contributions are listed in Section 8.4 and Section 8.5 details the joint academic contributions of this work. The concluding remarks of this dissertation

are described in Section 8.6.

## 8.1 Research Contributions

The main contributions of this work can be split into the following aspects: **i.** the realization of a systematic review on the field of software product lines evolution management; **ii.** the definition of a process to guide evolution in the product line context, in terms of change, build and release management in the core asset development level; **iii.** the definition of a process to guide evolution in the product line context in the product development level; **iv.** the execution of an experimental study which evaluated the early mentioned process. These contributions are further described next.

- **Systematic Review on Software Product Lines Evolution Management Approaches.** Through this review, seven approaches were identified and analyzed according to the aspects related to the systematic review's question: *How is evolution being managed in SPL?* The analysis results can be used to guide the definition of a new approach to SPL evolution management to core asset development and product development. In addition, the analysis results also can be useful to practitioners to choose the best approaches to evolution management.
- **RiPLE-EM for Core Asset Development.** After the systematic review, its results were the inputs to the definition of *RiPLE Evolution Management* process. RiPLE-EM is a process to guide evolution activities in terms of release management, change management and build management in the context of software product lines. RiPLE-EM for CAD is focused on the activities necessary to guarantee the proper evolution of core assets, from the moment they are planned and created, until the release of the core asset to the product development level.
- **RiPLE-EM for Product Development.** Consuming the core assets previously developed, instantiating some of the core assets to be customized, and creating product specific assets are common activities in the product development level. RiPLE-EM for PD goal is to manage the evolution of these products assets (instantiated, reused as it is, specific of the product) and the product itself in order to assure the software product lines benefits. Inside RiPLE-EM, there is also the concept of propagation request, which is the main gateway for changes to be propagated (feedback) from CAD to PD, or vice-versa.

- **The Experimental Study.** RiPLE-EM was used, in a formal experimental study, in an academic environment, to analyze its understandability, usability, completeness, applicability and effectiveness. This initial validation of RiPLE-EM helped in the improvement of the process, since the difficulties found by the subjects suggested some modifications in the activity flows related to the number of activity levels, which was reduced to improve understandability.

## 8.2 Related Work

Some evolution management approaches to SPL were identified through the systematic review, as described in Chapter 3. However, among the approaches included in this review, none of them had any kind of formal process definition for evolution management, and majority of them did not cover some disciplines in evolution management, such as change management and release management, indicating that the efforts on evolution management in SPL are still not much mature.

From the research community perspective, we could notice that there is still much to be explored. The output of the systematic review served as a foundation for the RiPLE-EM definition, which is focused on some of the questions raised in the systematic review. Some questions however were not taken in consideration in RiPLE-EM, and will be considered future work, as described as follows in the Section 8.3.

## 8.3 Future Work

Due to the time constraints imposed on a M.Sc. degree, this work can be seen as an initial step towards the efficient and effective evolution management in the context of software product lines. Thus, there are interesting topics to improve what was started, and new paths to explore. Thus, the following issues should be investigated as future work:

- **Metrics.** This dissertation proposed some metrics to evaluate RiPLE-EM use in the experimental study, however, these metrics were never used before, therefore they need to be refined and reproduced. This metric set could be also increased by several other metrics to measure software evolution in SPL.
- **Version Control and Continuous Integration.** Version control is an important topic inside evolution management, and there are many different strategies one could follow in a product line context. An important contribution would be a



process description guiding the engineer on versioning, branching, merging and tagging in a product line context. Moreover, *continuous integration* in the scope of product lines, could be explored for both core assets and products.

- **Configuration Identification.** If RiPLE-EM could give guidance on identifying core assets, structuring them, defining the appropriate baselines and setting acquisition rules, it would be an important contribution. There is not much information on configuration identification and what practical strategies to follow available in the literature, even for traditional configuration management.
- **Relationship between Product Line Architectures and Repository Structure.** The relationship between the Product Line Architecture and the configuration items and its structure in the repository is an important topic to future research inside the context of evolution management in SPL. Traditional systems do not handle structuring of items in the repository in a logical way (only physically), but establishing the relation between the SPL architecture and the items in the repository (both physically and logically structured) would be a interesting contribution.
- **Change Guidelines.** Another improvement for RiPLE-EM would be to have guidelines on how to change each kind of artifact (at least the most common ones) in a product line. These guidelines would have insights on *what*, *when* and *how* the different kinds of artifacts would be subject to change. Of course this improvement is constrained by the artifacts and technology diversity, but still the most common artifacts and the most common SPL technology could be addressed.
- **Variability Evolution.** Following the same line of thought, this future work item would be to have guidelines providing information about the evolution of variability points inside an asset. The addition, removal or change in the variability points may have great impacts (at least on every product that uses the asset being changed, and every other asset that depends on it) on the SPL, thus it needs to be well managed in order to avoid negative impacts.
- **Application of RiPLE-EM in an industrial context.** This dissertation presented the definition, planning, operation, analysis and interpretation of an experimental study. However, new studies are necessary in different contexts, with a more significant number of subjects, quantitative analysis in order to properly evaluate the use of RiPLE-EM. A very interesting context would be in a industrial/commercial context to verify if RiPLE-EM really fits these kind of contexts. In the preliminary

evaluation presented in this dissertation, RiPLE-EM already gives the impression that it would fit a industrial context with no bigger problems.

## 8.4 Academic Contributions

As a result of the work presented in this dissertation, the following contributions can be enumerated:

- (Anastasopoulos *et al.*, 2009) Evolving a Software Product Line Reuse Infrastructure: A Configuration Management Solution

Furthermore, the co-participation on the following publication contributed for acquiring experience and knowledge in the software product line and software reuse area:

- (de Souza Filho *et al.*, 2008) Evaluating Domain Design Approaches Using Systematic Review.

## 8.5 Joint Contributions

As a result of this work and the integration of this work with the Fraunhofer IESE, a joint paper was published, as described in the previous Section. This Section aims at detailing this joint publication, to provide more information about the partnership between this work and ther Fraunhofer IESE work.

It is important to remind that by the time of the publication (January, 2009) RiPLE-EM was slightly different than the version described and detailed in this dissertation (more specifically the Configuration Identification was removed from this final version).

The paper is entitled “Evolving a Software Product Line Reuse Infrastructure: A Configuration Management solution” and describes the interfaces between the RiPLE-EM and the Customization Layer (the tool developed at the Fraunhofer IESE by Michail Anastasopoulos) which is an automation layer that (a) defines basic product line evolution activities, (b) defines an asset model for the management of core asset / instance dependencies and (c) automates many of the manual steps that would be otherwise necessary if plain configuration management version control tools were used.

The solution is made up of two components, as shown in Figure 8.1, the RiPLE-EM and the Customization Layer (CL).

The Customization Layer is part of the PuLSE method (Product Line System and Software Engineering) and in particular it belongs to the PuLSE-EM technical component

---

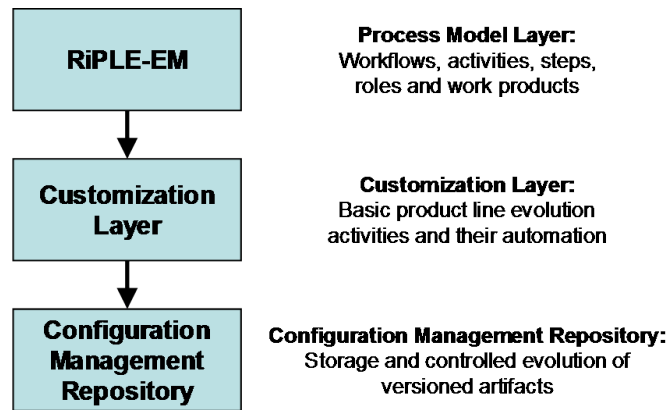
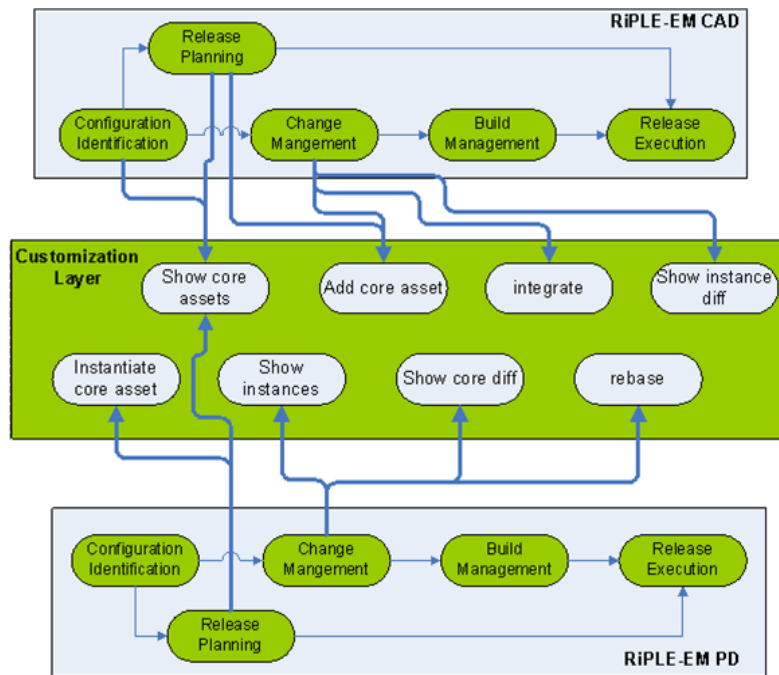


Figure 8.1 Solution Layers

(Evolution and Management), which also defines a process model similar to RiPLE-EM. The latter takes however a more configuration management-oriented view, by also addressing issues of Configuration Identification, Release and Build Management. Hence this paper aims at bringing together the RiPLE-EM and the PuLSE views.

The operations currently supported by the Customization Layer are the following. For simplicity the detailed signatures are left out at this point.

- **add-core-asset.** Create a core asset from an artifact (file or directory) and add it to the configuration management repository.
- **show-core-assets.** Given a location in the configuration management repository (or the location defined as standard if none is passed), it shows all core assets.
- **show-instance-diff.** Given a core asset, check whether its instances have changed since their derivation from the core asset.
- **integrate.** Given a core asset and one of its instances, mark the last change made to the core asset as a feedback from the instance to the core asset.
- **instantiate-core-asset.** Given a core asset create an instance of the core asset. The instance is basically a copy of the core asset where all or a part of the core asset variation is resolved. The Customization Layer however does not enforce resolving any variability. It simply creates a copy and assumes that some kind of development with reuse takes place during the copy operation or afterwards.
- **show-instances.** Given a core asset or the complete product line, show the current instances.



**Figure 8.2** RiPLE-EM and CL integration

- **show-core-diff.** Given an instance, check whether its core asset has changed since the derivation of the instance.
- **rebase.** Given an instance mark the last changes made to the instance as a feed-forward from the core asset of the instance to the instance itself.

To address a complete solution regarding evolution management in Software Product Lines, RiPLE-EM and the Customization Layer have integration points where both solutions can be combined to maximize the benefits. Each Customization Layer operation is triggered by a RiPLE-EM task or activity. The interfaces between the solutions are described next, and summarized in Figure 8.2.

- **add-core-asset.** This situation may occur in the change management flow, from both CAD and PD, when the need for a core asset creation is identified through change request analysis or through the configuration identification.
- **show-core-assets.** The need to visualize the available core assets may arise from the activity of identifying the configuration of a certain product which will re-use the core assets, from the instantiation of a core asset or from the propagation of changes.

- **show-instance-diff.** Any time a change propagation request is to be opened, it is important to know beforehand the changes made in the instance, in order to verify the applicability of the propagation. This operation is also triggered in the analysis of a propagation request.
- **integrate** Every time the propagation is realized, this operation is triggered to mark that the core asset was update with changes from a specific instance.
- **instantiate-core-asset.** When an instance needs to be created during PD configuration identification or PD change management this operation is triggered.
- **show-instances.** Given a certain core asset, it is always interesting to know which product have an instance of that asset, specially for the purpose of requesting propagation or simply analyzing a certain change request.
- **show-core-diff.** For product engineers, having the possibility of knowing when the base core asset for a given instance changed, to rebase it.
- **rebase.** Similar to the integrate operation. When the change propagation is realized, this operation is triggered to mark that the instance base was updated with changes from the core asset derived.

## 8.6 Concluding Remarks

Software reuse is a key aspect for organizations interested in achieving improvements in productivity, quality and costs reduction. Software product lines, as a software reuse approach, have proven its benefits in different industrial environments. Academic research in the software product line is also very rich, and a diversity of studies are being conducted in different topics of software product lines. Given this wide range of studies in the software product line field, this study tried to address a more specific issue in this area: *evolution management*. In this context, this dissertation presented RiPLE-EM process to evolution management. The process can be seen as a systematic way to guide and manage the evolution of every asset and product in a product line context, handling change management, build management and release management activities.

This dissertation also presented the initial validation of RiPLE-EM process, following well established guidelines to software experimentation. According to the data collected and analyzed in the experimental study, RiPLE-EM presents indications that the process can be viable.

Even it being an interesting contribution to the field, there were identified some new paths to explore in order to improve RiPLE-EM, such as the metrics associated with evolution in product lines, the relationship between product line architectures and repository structure, configuration identification, change guidelines to specific assets and new experimental studies in different contexts and environments.

Finally, we believe this dissertation is one more step to the maturation of the software evolution management in software product lines.

# Bibliography

- Almeida, E. S. (2007). *RiDE: The RiSE Process for Domain Engineering*. Ph.d. thesis, UFPE - Federal University of Pernambuco, Brazil. [1](#), [3.3.4](#), [8](#)
- Almeida, E. S., Alvaro, A., Lucrédio, D., Garcia, V. C., and de Lemos Meira, S. R. (2004). Rise project: Towards a robust framework for software reuse. In *IEEE International Conference on Information Reuse and Integration*, pages 48–53, Las Vegas, Las Vegas, NV, USA. [1.3.1](#)
- Almeida, E. S., Alvaro, A., Lucrédio, D., Garcia, V. C., and de Lemos Meira, S. R. (2005). A survey on software reuse processes. In *IEEE International Conference on Information Reuse and Integration*, pages 66–71, Las Vegas, Las Vegas, NV, USA. [1.3.1](#)
- Alvaro, A., de Almeida, E. S., and de Lemos Meira, S. R. (2006). A software component quality model: A preliminary evaluation. In *32nd EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 28–37, Cavtat, Dubrovnik, Croatia. [1.3.1](#)
- Anastasopoulos, M., de Oliveira, T. H. B., Muthig, D., de Almeida, E. S., and de Lemos Meira, S. R. (2009). Evolving a software product line reuse infrastructure: A configuration management solution. In *VaMoS - Variability Modelling of Software-intensive Systems*, pages 19–28, Sevilla, Spain. [1.1](#), [1.5](#), [5.3](#), [7.2.6](#), [8.4](#)
- Atkinson, C., Bayer, J., and Muthig, D. (2000). Component-based product line development: the kobra approach. In *International Software Product Line Conference*, pages 289–309, Denver, Colorado, United States. [2.2](#), [3.3.4](#), [3.4.4](#)
- 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, Boston, MA, USA. [1.1](#), [2.2](#), [2.3.2](#), [3.2.2](#), [3.3.4](#), [3.4.4](#), [4.2](#), [5.3](#)
- Basili, V. R., Caldiera, G., and Rombach, H. D. (1994). The goal question metric approach. In *Encyclopedia of Software Engineering*. Wiley. [7.1](#)
- Bay, M. E. (1999). *Software Release Methodology*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA. [3.2.2](#), [4.2](#)

- Bayer, J., Flege, O., Knauber, P., Laqua, R., Muthig, D., Schmid, K., Widen, T., and DeBaud, J.-M. (1999). Pulse: a methodology to develop software product lines. In *SSR - Symposium on Software Reusability*, pages 122–131, New York, NY, USA. 1.1, 3.3.4, 8
- Berczuk, S. P. and Appleton, B. (2002). *Software Configuration Management Patterns: Effective Teamwork, Practical Integration*. Addison-Wesley, Boston, MA, USA. 5.3
- Beuche, D. (2008). Modeling and building software product lines with pure::variants. In *International Software Product Line Conference*, page 358, Washington, DC, USA. 3.4.5, 4.5, 5.4, 6.4
- Brereton, P. (1999). Evolution of component based systems. In *International Conference on Software Engineering, Component-Based Software Engineering Workshop*, Los Angeles, CA, USA. 1
- Brereton, P., Kitchenham, B. A., Budgen, D., Turner, M., and Khalil, M. (2007). Lessons from applying the systematic literature review process within the software engineering domain. *Journal of Systems and Software*, **80**(4), 571–583. 3.3.3, 3.3.6
- Budgen, D. and Brereton, P. (2006). Performing systematic literature reviews in software engineering. In *International conference on Software engineering*, pages 1051–1052, Shanghai, China. 3.3.4
- Burégio, V. (2006). *Specification, Design and Implementation of Reuse Repository*. M.sc. dissertation, UFPE - Federal University of Pernambuco, Brazil. 1.3.1
- Cavalcanti, Y. C., Martins, A. C., Almeida, E. S., and Meira, S. R. L. (2008). Avoiding duplicate cr reports in open source software projects. In *The 9th International Free Software Forum (IFSF'08)*, Porto Alegre, Brazil. 1.3.1
- Chrissis, M. B., Konrad, M., and Shrum, S. (2003). *CMMI: Guidelines for Process Integration and Product Improvement*. Addison-Wesley. 3.4.1
- Clements, P. and Northrop, L. (2002). *Software Product Lines: Practices and Patterns*. Addison-Wesley. (document), 1.1, 2.1, 2.1.1, 2.1.1, 2.1, 2.1.1, 2.2, 2.3, 2.4
- Cook, J. E. and Dage, J. A. (1999). Highly reliable upgrading of components. In *International Conference on Software Engineering*, pages 203–212, Los Angeles, CA, USA. 1
-



- Crnkovic, I. and Larsson, M. (1999). Managing standard components in large software systems. In *Second International Workshop on Component-Based Software Engineering*, Los Angeles, CA, USA. [1](#)
- de Souza Filho, E. D., de Oliveira Cavalcanti, R., Neiva, D. F. S., de Oliveira, T. H. B., Lisboa, L. B., de Almeida, E. S., and de Lemos Meira, S. R. (2008). Evaluating domain design approaches using systematic review. In *European Conference on Software Architecture*, pages 50–65, Paphos, Cyprus. [8.4](#)
- dos Santos Brito, K., Garcia, V. C., de Almeida, E. S., and de Lemos Meira, S. R. (2008). Lift - a legacy information retrieval tool. *Journal of Universal Computer Science*, **14**(8), 1256–1284. [1.3.1](#)
- Durao, F. A. (2008). *Semantic Layer Applied to a Source Code Search Engine*. M.sc. dissertation, UFPE - Federal University of Pernambuco, Brazil. [1.3.1](#)
- Fuggetta, A. (2000). Software process: a roadmap. In *International Conference on Software Engineering*, pages 25–34, Limerick, Ireland. [3.2.2](#)
- Gacek, C. and Anastasopoulos, M. (2001). Implementing product line variabilities. *SIGSOFT Software Engineering Notes*, **26**(3), 109–117. [3.2.2](#), [4.2](#)
- Garcia, V. C., Lisboa, L. B., de Lemos Meira, S. R., de Almeida, E. S., Lucrédio, D., and de Mattos Fortes, R. P. (2008). Towards an assessment method for software reuse capability. In *International Conference on Quality Software*, pages 294–299, Oxford, UK. [1.3.1](#)
- Genssler, T., Christoph, A., Winter, M., Nierstrasz, O., Ducasse, S., Wuyts, R., Arévalo, G., Schönhage, B., Müller, P. O., and Stich, C. (2002). Components for embedded software: the pecos approach. In *Conference on Compilers, Architectures and Synthesis for Embedded Systems*, pages 19–26, Grenoble, France. [3.3.4](#)
- Gomaa, H. and Shin, M. E. (2002). Multiple-view meta-modeling of software product lines. In *International Conference on Engineering of Complex Computer Systems*, page 238, Washington, DC, USA. [3.3.4](#)
- Kitchenham, B. (2007). Guidelines for performing systematic literature reviews in software engineering. Technical report, Keele University and NICTA. [3](#), [3.1](#), [3.2](#), [3.2.4](#), [3.3](#)
-

- Krueger, C. W. (2002). Variation management for software production lines. In *International Software Product Line Conference*, pages 37–48, London, UK. (document), 1.1, 2.1.2, 8
- Krueger, C. W. (2008). The biglever software gears unified software product line engineering framework. In *International Software Product Line Conference*, page 353, Washington, DC, USA. 3.4.5, 4.5, 5.4, 6.4
- Kurmann, R. (2006). Agile software product line configuration and release management. In *Workshop on Agile Product Line Engineering in the Software Product Line Conference*, Baltimore, Maryland, USA. 3.3.4, 3.4.3, 3
- Lehman, M., Ramil, J., and Kahen, G. (2000). Evolution as a noun and evolution as a verb. In *Workshop on Software and Organisation Co-evolution*, Imperial College, London. 2.2.2
- Lehman, M. M. (1980). On understanding laws, evolution, and conservation in the large-program life cycle. *Journal of Systems and Software*, 1, 213–221. 2.2, 2.2.1
- Lientz, B. P. and Swanson, E. B. (1980). *Software Maintenance Management*. Addison-Wesley, Boston, MA, USA. 2.2.2
- Lisboa, L. B. (2008). *ToolDAy - A Tool for Domain Analysis*. M.sc. dissertation, UFPE - Federal University of Pernambuco, Brazil. 1.3.1
- Martins, A. C., Garcia, V. C., Almeida, E. S., and Meira, S. R. L. (2008). Enhancing components search in a reuse environment using discovered knowledge techniques. In *2nd Brazilian Symposium on Software Components, Architectures, and Reuse*. 1.3.1
- Mascena, J. C. C. P., de Lemos Meira, S. R., de Almeida, E. S., and Garcia, V. C. (2006). Towards an effective integrated reuse environment. In *International Conference on Generative Programming and Component Engineering*, pages 95–100, Portland, Oregon, USA. 1.3.1
- McGregor, J. D. (2003). Evolution of product line assets. Technical report, Software Engineering Institute. (document), 1.1, 2.3, 2.3.1, 3.3.4, 3.4.4, 8
- McGregor, J. D. (2007). Cm - configuration change management. *Journal of Object Technology*, 6(1), 7–15. 1, 3.3.4, 3.4.4, 4.2
-

- Mendes, R. C. (2008). *Search and Retrieval of Reusable Source Code using Faceted Classification Approach*. M.sc. dissertation, UFPE - Federal University of Pernambuco, Brazil. [1.3.1](#)
- Mens, T. and Demeyer, S. (2008). *Software Evolution*. Springer. [2.2](#)
- Mohan, K. and Ramesh, B. (2006). Change management patterns in software product lines. *Communications of the ACM*, **49**(12), 68–72. [3.3.4](#), [3.4.4](#)
- Mohan, K., Xu, P., and Ramesh, B. (2008). Improving the change-management process. *Communications of the ACM*, **51**(5), 59–64. [3.3.4](#)
- Naur, P. and Randell, B. (1969). *Software Engineering: Report of a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7-11 Oct. 1968*. Scientific Affairs Division, NATO, Brüssel. [2](#), [2.2](#)
- Northrop, L. M. (2002). Sei’s software product line tenets. *IEEE Softw.*, **19**(4), 32–40. [2.1](#)
- Pohl, K., Böckle, G., and van der Linden, F. J. (2005a). *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag, Secaucus, NJ, USA. [1.1](#), [2.1.2](#), [2.1.3](#)
- Pohl, K., Böckle, G., and van der Linden, F. J. (2005b). *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag, Secaucus, NJ, USA. [1.1](#), [2.1](#), [2.1.3](#)
- Pussinen, M. (2002). A survey on software product-line evolution. Technical report, Institute of Software Systems, Tampere University of Technology. [\(document\)](#), [2.3](#), [3.6](#)
- Royce, W. (1970). Managing the development of large software systems. In *International Conference on Software Engineering*, pages 1–9, Monterey, California, United States. [2.2](#)
- Santos, E. C. R., Durão, F. A., Martins, A. C., Mendes, R., Melo, C. d. A., Garcia, V. C., Almeida, E. S., and Meira, S. R. d. L. (2006). Towards an effective context-aware proactive asset search and retrieval tool. In *Sixth Workshop on Component-Based Development*, pages 105–112, Recife, Brazil. [1.3.1](#)
- SPEM, OMG (2008). Software Process Engineering Metamodel (SPEM). Technical report, Object Management Group. [4.9](#), [4.9.1](#)
-

- Staples, M. (2004). Change control for product line software engineering. In *Asia-Pacific Software Engineering Conference*, pages 572–573, Washington, DC, USA. 3.3.4, 3.4.3, 3.4.4
- SWEBOK (2007). Software engineering body of knowledge. Technical report, Fraunhofer IESE, Robert Bosch GmbH, University of Groningen, University of Karlskrona/Ronneby, Siemens. 3.2.2
- Taborda, L. J. M. (2003). Planning and managing product line evolution. In *International Workshop on Product Family Engineering*, Siena, Italy. 3.3.4, 3.4.3
- van der Hoek, A. and Wolf, A. L. (2003). Software release management for component-based software. *Software Practice and Experience*, **33**(1), 77–98. 2.2, 3.3.4, 3.4.3
- van der Hoek, A., Hall, R. S., Heimbigner, D., and Wolf, A. L. (1997). Software release management. In *European Software Engineering Conference*, pages 159–175, Zurich, Switzerland. 3.2.2, 3.3.4, 4.2
- van Ommering, R., van der Linden, F., Kramer, J., and Magee, J. (2000). The koala component model for consumer electronics software. *Computer*, **33**(3), 78–85. 3.3.4
- van Ommering, R. C. (2000). Beyond product families: Building a product population? In *International Workshop on Software Architectures for Product Families*, pages 187–198, London, UK. 3.2.2, 3.4.2
- Vanderlei, T. A., Durão, F. A., Martins, A. C., Garcia, V. C., de Almeida, E. S., and de Lemos Meira, S. R. (2007). A cooperative classification mechanism for search and retrieval software components. In *ACM Symposium on Applied Computing*, pages 866–871, Seoul, Korea. 1.3.1
- Vegas, S. and Basili, V. (2005). A characterisation schema for software testing techniques. *Empirical Software Engineering*, **10**(4), 437–466. 7.1.3
- Voas, J. (1998). Maintaining component-based systems. *IEEE Software*, **15**(4), 22–27. 1, 5.3
- Weiss, D. M. and Lai, C. T. R. (1999). *Software Product-Line Engineering: A Family-Based Software Development Process*. Addison-Wesley. 1.1, 3.3.4, 8
-

- Weiss, D. M., Clements, P. C., Kang, K., and Krueger, C. (2006). Software product line hall of fame. In *International Software Product Line Conference*, page 237, Washington, DC, USA. (document), 1, 2.4, 8
- Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., and Wesslén, A. (2000). *Experimentation in software engineering: an introduction*. Kluwer Academic Publishers, Norwell, MA, USA. (document), 7, 7.1, 7.1.3, 7.2.1, 7.2.4, 7.2.5, 7.2.7
- Yau, S. S., Collofello, J. S., and MacGregor, T. M. (1993). Ripple effect analysis of software maintenance. pages 71–82. 2.2

# Appendices



# Experiment Questionnaires

As part of the experiment instrumentation, detailed in Chapter 7, two questionnaires were defined, and applied to the subjects. The next sections lists all questions inside each questionnaire.

The first questionnaire (detailed in Section A.1) was intended to collect data about the subjects background, and the second one (detailed in Section A.2) was intended to collect information about RiPLE-EM use.

## A.1 QT1 - Background Questionnaire

**Date:**

**Name:**

**Amount of Years since graduation:**

1. How many commercial software projects did you participate after graduation?

Category	Quantity
Low Complexity:	
Medium Complexity:	
High Complexity:	

2. About your personal experience in Evolution (change management, release management, build management, configuration management) (mark x):

Area	None	Low	Medium	High	Years
Academic					
Commercial					

Area	None	Low	Medium	High	Years
Academic					
Commercial					

3. About your personal experience in software reuse (mark x):
4. Please, inform which techniques/methods you know in Evolution, SCM and software reuse areas

## A.2 QT2 - RiPLE-EM Analysis Questionnaire

Date:

**Regarding the evolution management using the RiPLE-EM process, please answer the following questions:**

1. Did you have any difficulties in understanding or applying the evolution management process (RiPLE-EM)? Which one(s)?
2. In your opinion, what are the strengths of the RiPLE-EM process?
3. In your opinion, what are the weak points and bottlenecks of the RiPLE-EM process?
4. Is there any missing activity, roles or artifact in the RiPLE-EM process?
5. Which improvements would you suggest for the evolution management activities?
6. What is your satisfaction in to use the RiPLE-EM process?

( ) Very satisfied ( ) Satisfied ( ) Impartial ( ) Unsatisfied ( ) Very Unsatisfied



# B

## RiPLE-EM Checklists and Templates

In this appendix, all checklists and templates of RiPLE-EM will be described and detailed.

### B.1 Change Request Analysis Checklist

Every change request must be analyzed before implemented. The group responsible for the analysis is called CCB (Change Control Board), and the CCB can be formed by different roles and persons.

The purpose of this analysis, among others, is:

- Priorization of Changes.
- Identification of each change's impacts.
- Identify similarities between the changes requested and Group changes regarding its similarity.
- Guarantee that changes feedbacks are being implemented.

Based on these purposes, the change analysis must be performed following a specific checklist of what to analyze and what to do in order to be effective in the change analysis and not to spend too much time on this analysis.

Ideally, the analysis made has to be documented somewhere, and one of the best places to document it is inside the CR being analyzed. The CCB is then responsible to generate the documentation of the analysis following the template defined and post it in the CR it can be reviewed in the future (in case of similar changes) and may serve as the rationale for each changes implementation.

Some of the following checks were inspired in some checks of the *process impact*<sup>1</sup>

---

<sup>1</sup>Process Impact web site - <http://www.processimpact.com>

site.

The checks are listed next.

### **B.1.1 Checks**

- Identify any existing requirements in the baseline that conflict with the proposed change.
- Identify any other pending requirement changes that conflict with the proposed change.
- What are the consequences of not making the change?
- What are the possible side effects or other risks of making the proposed change?
- Is the proposed change feasible within known technical constraints and current staff skills?
- Identify any third party software that must be purchased.
- How will the proposed change affect the sequence, dependencies, effort, or duration of any tasks currently in the project schedule?
- Will prototyping or other user input be required to verify the proposed change?
- How much effort that has already been invested in the project will be lost if this change is accepted?
- Identify any user interface changes, additions, or deletions required.
- Identify any changes, additions, or deletions required in reports, databases, or data files.
- Identify the design components that must be created, modified, or deleted.
- Identify hardware components that must be added, altered, or deleted.
- Identify any changes required in build files.
- Identify existing unit, integration, system, and acceptance test cases that must be modified or deleted.

- Identify any help screens, user manuals, training materials, or other documentation that must be created or modified.
- Identify any other systems, applications, libraries, or hardware components affected by the change.
- Identify any impact the proposed change will have on fielded systems if the affected component is not perfectly backward compatible.

## **B.2 Propagation Request Analysis Checklist**

Every time a propagation is going to be performed, the task called "Analyze Propagation" should be prior performed. The purpose of this analysis is to verify if there no problems regarding the propagation of the changes. In the propagation analysis, some items need to be checked and for that follow the checks in this page.

Besides all CR checks, the PR may have the following checks.

### **B.2.1 Checks**

- Verify test results of the version (change set) being propagated?
- Check compatibility to see if the changes targeted to be propagated support the product being developed.
- Identify the type of the propagation

## **B.3 CCB Analysis Template**

In order to document all analysis done to both a CR or a PR, a minimum CCB analysis documentation template is proposed.

### **B.3.1 Template**

- CCB participants:
  - Participants: <one>, <two>, <three>
- Decisions and Rationale:

- Decisions:
- Rationales:
- Identified Impacts:
  - Core Assets Impacts:
  - Products Impacts:

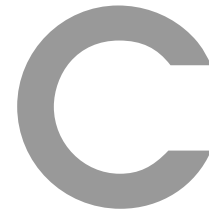
Examples:

1)

- CCB participants:
  - Participants: Thiago Burgos, Ednaldo Filho, Danuza Neiva
- Decisions and Rationale:
  - Decisions: This CR will be aborted.
  - Rationales: This is not an error, but an expected behavior.
- Identified Impacts:
  - Core Assets Impacts: None
  - Products Impacts: None

2)

- CCB participants:
  - Participants: Thiago Burgos
- Decisions and Rationale:
  - Decisions: This CR fix has to be included in next release.
  - Rationales: N/A
- Identified Impacts:
  - Core Assets Impacts: Core Asset Requirements, Use Cases and Test Cases.
  - Products Impacts: None



# Systematic Review Sources

This appendix lists the conferences, journals and web-search engines used to search for primary studies candidates in the systematic review procedure executed and detailed in Chapter 3.

## C.1 Journals Sources

All journals searched for primary studies are listed next:

- **IEEE Software** - <http://www.computer.org/portal/site/software/>
- **IEEE Transactions on Software Engineering** - <http://www.computer.org/tse/>
- **IEEE Computer** - <http://www.computer.org/portal/site/computer/index.jsp>
- **IEEE ACM SIGSOFT Software Engineering Notes** - <http://www.sigsoft.org/SEN/>
- **Communications of the ACM** - <http://cacm.acm.org/>
- **Journal of Systems and Software** - <http://www.elsevier.com/locate/jss>
- **Journal of Software Maintenance: Research and Practice** - <http://www.informatik.uni-trier.de/ley/db/journals/smr/index.htm>
- **International Journal of Information Systems and Change Management** - <http://www.informatik.uni-trier.de/ley/db/journals/ijiscm/index.html>

## C.2 Conference Sources

All conferences searched for primary studies are listed next:

- **International Conference on Software Engineering (ICSE)** - <http://www.icse-conferences.org/>
- **Fundamental Approaches to Software Engineering (FASE)** - <http://www.informatik.uni-trier.de/ley/db/conf/fase/index.html>
- **International Conference on Software Reuse (ICSR)** - <http://www.informatik.uni-trier.de/ley/db/conf/icsr/index.html>
- **Software Product Line Conference (SPLC)** - <http://splc.net/>
- **European Software Engineering Conference (ESEC)** - <http://www.informatik.uni-trier.de/ley/db/conf/esec/index.html>
- **International Computer Software and Applications Conference (COMPSAC)**  
- <http://wotan.liu.edu/docis/dbl/compsa/index.html>
- **International Conference on Software Maintenance (ICSM)** - <http://conferences.computer.org/icsm/>
- **Software Product Family Engineering Conference (PFE)** - <http://www.informatik.uni-trier.de/ley/db/conf/pfe/index.html>
- **European Conference on Software Maintenance and Re-engineering (CSMR)**  
- <http://www.csmr.eu/>
- **International Conference on Program Comprehension (ICPC)** - <http://www.program-comprehension.org/>
- **International Conference on Software Engineering and Knowledge Engineering (SEKE)** - <http://www.informatik.uni-trier.de/ley/db/conf/seke/index.html>
- **International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)** - <http://www.oopsla.org/>

## C.3 Web-Search Engine Sources

All web-search engines used to search for primary studies are listed next:

- **Google scholar** - <http://scholar.google.com>
- **Citeseer** - <http://citeseer.ist.psu.edu>
- **Citeulike** - <http://www.citeulike.com>