computação
UFBA

“TITLE”

By

Karla Malta Amorim da Silva

B.Sc. Dissertation

Federal University of Bahia
ceagmat@ufba.br
wiki.dcc.ufba.br/DCC/

SALVADOR, November/2015

Federal University of Bahia
Computer Science Department
Bachelor's Degree in Computer Science

Karla Malta Amorim da Silva

"TITLE"

A B.Sc. Dissertation presented to the Computer Science Department of Federal University of Bahia in partial fulfillment of the requirements for the degree of Bachelor in Computer Science.

Advisor: *Eduardo Santana de Almeida*
Co-Advisor: *Raphael Pereira de Oliveira*

SALVADOR, November/2015

I dedicate this dissertation to my family, friends and professors who gave me all necessary support to get here.

Blessed is the man who finds wisdom, the man who gains understanding.

—PROVERBS 3:13

Resumo

A engenharia de software é uma disciplina que integra processos, métodos e ferramentas para o desenvolvimento de programas de computador. Para apoiá-la existem sistemas de gerenciamento de ciclo de vida de aplicativos (ALM), que são ferramentas de software que auxiliam na organização e moderação de um projeto ao longo de seu ciclo de vida. Existe um número infinito de possibilidades para definir o ciclo de vida de um projeto, e as ferramentas necessárias para ajudá-la a conclusão. Consequentemente, uma enorme variedade de sistemas de gestão estão disponíveis no mercado, especificamente concebidos de acordo com um número diversificado de metodologias de gestão. No entanto, a maioria dos sistemas são proprietários, bastante especializados e com pouca ou nenhuma capacidade de personalização, tornando muito difícil encontrar um que se encaixam perfeitamente às necessidades do seu projeto.

Um tipo diferente de desenvolvimento de sistemas de software é a Engenharia de Linha de Produtos de Software - ELPS. LPS é uma metodologia para o desenvolvimento de uma diversidade de produtos de software relacionados e sistemas com uso intensivo de software. Durante o desenvolvimento de uma LPS, uma vasta gama de artefatos devem ser criados e mantidos para preservar a consistência do modelo de família durante o desenvolvimento, o e que é importante para controlar a variabilidade SPL e a rastreabilidade entre os artefatos. No entanto, esta é uma tarefa difícil, devido à heterogeneidade dos artefatos desenvolvidos durante engenharia de linha de produto. Manter a rastreabilidade e artefatos atualizados manualmente é um processo passível de erro, demorado e complexo. Utilizar uma ferramenta para apoiar essas atividades é essencial.

Neste trabalho, propomos o Ambiente Construção Integrado de Linha de Produto de Software (SPLICE). Essa é uma ferramenta online de gerenciamento de ciclo de vida que gerencia, de forma automatizada, as atividades da linha de produtos de software. Esta iniciativa pretende apoiar a maior parte das atividades do processo de LPS, como escopo, requisitos, arquitetura, testes, controle de versão, evolução, gestão e práticas ágeis.

Nós apresentamos um metamodelo leve, que integra o processo de ciclo de vida das Linhas de Produtos com práticas ágeis, a implementação de uma ferramenta que utiliza o metamodelo proposto , e um estudo de caso que reflete a viabilidade e flexibilidade desta solução especialmente para diferentes cenários e processos.

Palavras-chave: ferramenta, busca linha de produtos de software, métodos ágeis, LPS , sistema de gerenciamento de ciclo de vida de aplicativos , ferramenta, metamodelo

Abstract

Software engineering is a discipline that integrates process, methods, and tools for the development of computer software. To support it, Application lifecycle management systems are software tools that assist in the organization and moderation of a project throughout its life cycle. There is an infinite number of possibilities to define a project life cycle, and the needed tools to help it completion. Consequently, a huge variety of management systems are available on today's market, specifically designed to conform to a diverse number of management methodologies. Nevertheless, most are proprietary, very specialized, with little to no customization, making very hard to find one that perfectly fit one's needs.

A different type of software systems development is Software Product Line Engineering – SPLE. SPL is a methodology for developing a diversity of related software products and software-intensive systems. During the development of a SPL, a wide range of artifacts needs to be created and maintained to preserve the consistency of the family model during development, and it is important to manage the SPL variability and the traceability among those artifacts. However, this is a hard task, due to the heterogeneity of assets developed during product line engineering. Maintaining the traceability and artifacts updated manually is error-prone, time consuming and complex. Utilizing a tool for supporting those activities is essential.

In this work, we propose the Software Product Line Integrated Construction Environment (SPLICE). That is a web-based life cycle management tool for managing, in an automated way, the software product line activities. This initiative intends to support most of the SPL process activities such as scoping, requirements, architecture, testing, version control, evolution, management and agile practices. This was archived with the integration of a framework around an established tool, providing an easy way for handling the usage of different metamodels.

We present a lightweight metamodel which integrates the processes of the SPL lifecycle agile practices, the implementation of a tool that uses the proposed metamodel, and a case study that reflect the feasibility and flexibility of this solution especially for different scenarios and processes

Keywords: software product line, agile, SPL, Application lifecycle management , tool, metamodel

Contents

List of Figures

List of Tables

List of Acronyms

ALM	Application Lifecycle Management
AP	Agile Planning
BTT	Bug Report Tracker Tool
C.E.S.A.R.	Recife Center For Advanced Studies and Systems
CAD	Core Asset Development
CASE	Computer-Aided Software Engineering
CRUD	Create, read, update and delete
FeDRE	Feature-Driven Requirements Engineering Approach
RE	Requirements Engineering
RiSE	Reuse in Software Engineering
RQ	Requirements
ORM	Object-relational mapping
PD	Product Development
SPL	Software Product Line
SPLE	Software Product Line Engineering
SE	Software Engineering
SW	Software
SC	Scoping
SPLICE	Software Product Line Integrated Construction Environment
SWIG	Simplified Wrapper and Interface Generator
TE	Tests
OT	Others

VCS Version control systems

VM Variability Management

1 Introduction

A Software Product Line (SPL) is outlined as a collection of similar software intensive systems that share a set of common features satisfying the wants of specific customers, market segments or mission. Those similar software systems are developed from a set of core assets, comprised of documents, specifications, components, and other software artifacts that may be reusable throughout the development of each system within the product line (Capilla *et al.*, 2013).

Requirements are typical assets in SPL. They are specified in reusable models, in which commonalities and variabilities are documented explicitly. Thus, these requirements can be instantiated and adapted to derive the requirements for an individual product (Cheng and Atlee, 2007). New products in the SPL will be much simpler to specify, because the requirements are reused and tailored (Clements and Northrop, 2002).

Requirements Engineering (RE) in SPL has an additional cost. Many SPL requirements are complex, interlinked, and divided into common, variable and product-specific requirements (Birk *et al.*, 2003; de Oliveira *et al.*, 2014). The requirements engineering process must be tool-supported to handle complexity and the huge volume of elicited requirements (Birk *et al.*, 2003).

The focus of this dissertation is to provide a support tool for performing the specification of the SPL requirements in a systematic way through the use of guidelines, showing step by step how the specification should be done.

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 aspects that are not directly addressed by this work. Section 1.5 presents the main contributions, Section 1.6 presents the research design and, finally, Section 1.7 outlines the structure of this dissertation.

1.1 Motivation

1.2 Problem Statement

1.3 Related Work

In order to accomplish the goal of this dissertation, we propose the Software Product Line Integrated Construction Environment (SPLICE). This tool supports the Software Product Line (SPL) process activities in order to assist engineers in the traceability, variability management and maintenance activity. The remainder of this section presents the context where it was developed and the outline of the proposed solution.

1.3.1 Context

This dissertation describes a tool that is part of the Reuse in Software Engineering (RiSE) (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. RiSE Labs it is influenced by a series of areas, such as software measurement, architecture, quality, environments and tools, and so on, in order to achieve its goal. The influence areas can be seen 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, this framework embraces several different projects related to software reuse and software engineering. They are:

- **RiSE Framework:** Involves reuse processes (Almeida *et al.*, 2004; Nascimento, 2008), 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, 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 (Durao, 2008), facets (Mendes, 2008) and data mining (Martins *et al.*, 2008), and the Legacy InFormation retrieval Tool (LIFT) (Brito, 2007), the Reuse Repository System (CORE) (Melo, 2008), and the Tool for Domain Analysis (ToolDAy) (Lisboa, 2008). This dissertation is part of the RiSE tools;

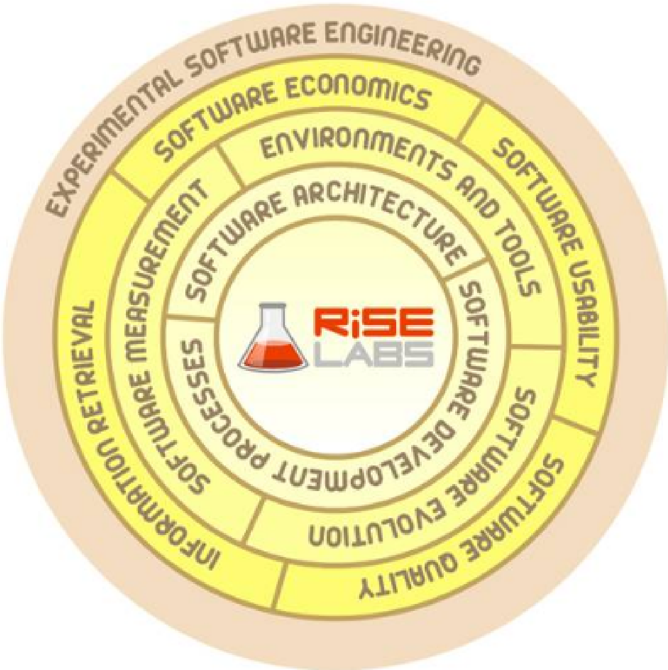

Figure 1.1 RiSE Labs Influences

- **RiPLE:** Stands for RiSE Product Line Engineering Process and aims at developing a methodology for Software Product Lines, composed of scoping (Moraes, 2010), requirements engineering (Neiva, 2009), design (de Souza Filho, 2010; Cavalcanti *et al.*, 2011) , implementation, test (da Mota Silveira Neto, 2010; do Carmo Machado, 2010), and evolution management (de Oliveira, 2009).

- **SOPLE:** Development of a methodology for Service-Oriented Product Lines, based on the fundamentals of the RiPLE (Ribeiro, 2010).

- **MATRIX:** Investigates the area of measurement in reuse and its impact on quality and productivity;

- **BTT:** Research focused on tools for detection of duplicate bug reports, such as in Cavalcanti *et al.* (2008, 2012).

- **Exploratory Research:** Investigates new research directions in software engineering and its impact on reuse;

- **CX-Ray:** Focused on understanding the Recife Center For Advanced Studies and Systems (C.E.S.A.R.), and its processes and practices in software development.

This dissertation is part of the RiSE Tools project. It was conducted in collaboration with researchers in software reuse , to solve the problem of traceability during the life-cycle of aSoftware Product Line (SPL) development.

Figure 1.2 RiSE Labs Projects

1.3.2 Outline of the Proposal

This work defines the requirements, design and implementation of a software product lines lifecycle management tool, providing traceability and variability management and supporting most of the SPL process activities such as scoping, requirements, architecture, testing, version control, evolution, management and agile practices. In order to address it, we propose a metamodel that covers the SPL lifecycle, and develop a solution that consists in a Web based, extensible SPL lifecycle management tool, implementing this metamodel.

The tool must enable the engineers involved in the process, to automatize the assets creation and maintenance, while providing traceability and variability management between them, providing detailed reports and enable the engineers to easily navigate between the assets using the traceability links. It must also provide a basic infrastructure for development, and a centralized point for user management among different tools.

1.4 Out of Scope

The following topics are not considered in the scope of this dissertation:

- **SPL Domain Requirements Evolution**

  Although an approach has already been proposed for the SPL domain requirements evolution phase (FeDRE2), we still do not support this approach, but it is certainly a direction we intend to follow in the future.

- **SPL Application Requirements Engineering**

  In this work we do not consider the SPL Application Engineering process, then our contributions do not cover the SPL Application Requirements Engineering.

- **Non-SPL Tools**

  This work is concerned with Software Product Lines development and tools and environments that support the SPL approach. Non-SPL tools are out of scope.

1.5 Statement of the Contributions

As a result of the work presented in this dissertation, the following contribution can be highlighted:

- **Tool support for a SPL domain requirements specification approach (FeDRE)** We extended the SPLICE tool, a SPL lifecycle management tool and automated Feature-Driven Requirements Engineering Approach (FeDRE), thus improving the automation of Software Product Lines (SPL) requirements engineering phase.

1.6 Research Design

The first step of our work was to investigate the software product line area. This informal study also included to understand the requirements engineering phase for single systems and software product lines. As a result, we could write out the second chapter with some foundations on these subjects.

During the informal study we identified the need for tools that appropriately support the domain requirements engineering phase of software product lines. After choosing a requirements specification approach (FeDRE), we extended an existing SPL lifecycle management tool (SPLICE) providing tool support for this approach.

In order to evaluate the proposed tool, we conducted a survey to identify limitations and needed improvements for the tool.

1.7 Dissertation Structure

The remainder of this dissertation is organized as follows:

- **Chapter 2** reviews the essential topics related to this work: Software Product Lines SPL; requirements engineering; SPL requirements engineering; and Software Product Line Engineering (SPLE) tool support.

- **Chapter ??** describes the SPLICE tool, its architeture and the set of frameworks and technologies used during its development. Also, presents the new functional and non-functional requirements proposed for FeDRE implementation based upon SPLICE.

- **Chapter ??** describes an evaluation of FeDRE implementation.

- **Chapter ??** provides the concluding remarks. It discusses our contributions, limitations, threats to validity, and outlines directions for future work.

2

An Overview on Software Product Lines, Requirements Engineering, SPL Requirements Engineering and SPLE Tool Support

This chapter presents fundamental information for the understanding of four topics that are relevant to this work: software product lines, requirements engineering, and SPL requirements engineering. Section 2.1 discusses the motivation, benefits, and the SPL development process. Section 2.2 presents requirements engineering. Section 2.3 presents SPL requirements engineering. Section 2.4 presents SPLE Tool Support. Finally, Section 2.5 presents a summary of this chapter.

2.1 Software Product Lines

2.1.1 Introduction

Nowadays we experience the age of customization, but it was not always like that. There was a time when goods were handcrafted for individual costumers. Over the years, the number of people who could afford to buy several kinds of products has increased (Pohl *et al.*, 2005). In order to meet this rising demand, the production line was invented, which enabled production for a mass market much more cheaply than individual product.

Customers were satisfied with mass produced products for a while (Pohl *et al.*, 2005), however that kind of product lacks sufficient diversification to meet individual customers' wishes. Individualized products also have a drawback; they are a lot more expensive than standardized products. In that context, the industry was challenged to provide customized products at reasonable costs to satisfy the wishes of specific customers and market segments. The combination of mass customization and common platforms was the key to achieve that goal.

Mass customization is the large-scale production of goods tailored to individual customers' needs. It requires a higher technological investment which leads to higher prices for the individualized products and/or to lower profit margins for the company. The platform approach though, enables manufacturers to offer a larger variety of products and to reduce costs at the same time. A platform is defined as a base of technologies on which other technologies or processes are built. The combination of mass customization and a common platform allows us to reuse a common base of technology and to bring out products in close accordance with customers' wishes (Pohl *et al.*, 2005).

In the software domain, that combination resulted in a software development paradigm called Software Product Line Engineering (SPLE). A Software Product Line (SPL) is a set of software-intensive systems that share a common, managed feature set, satisfying a particular market segment's specic needs or mission and that are developed from a common set of core assets in a prescribed way (Clements and Northrop, 2002).

2.1.2 The Benefits

Developing software under the Product Line Engineering paradigm offers many benefits for a company, some examples follow:

- **Reduction of Development Costs**

  A good reason for applying the Product Line Engineering paradigm is the reduction of costs as the reuse of assets increases. Through the reuse of artifacts from the platform in different systems, the development of each of these systems becomes cheaper. First, the company has to invest in the development of the platform. Also, the way in which the artefacts from the platform will be reused has to be well planned beforehand. Then, from a certain point, called break-even point, the initial investment will be paid off. The precise location of this point is influenced by many characteristics of the company, the market it has envisaged, its customers, expertise, kinds of products, the way the product line is created and others.

  Figure 2.1 shows that the costs to develop a few systems in an SPL approach are higher than in a single systems approach. However, using product line engineering, the costs are significantly lower for larger systems quantities.

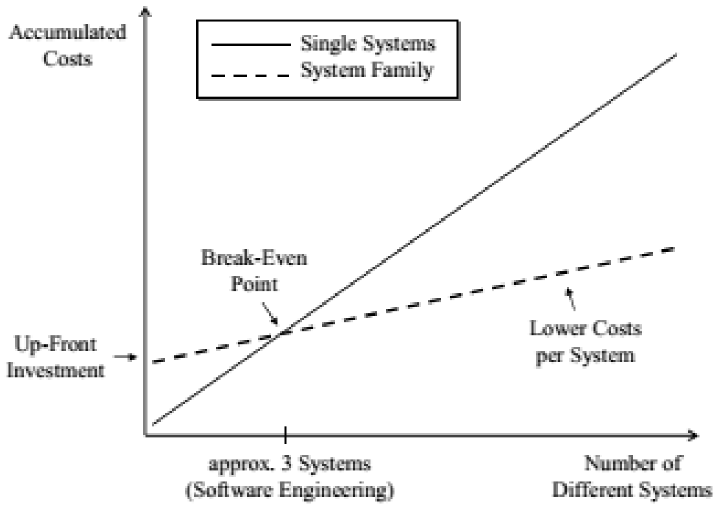

Figure 2.1 Costs for developing systems as single systems compared to product line engineering (Pohl *et al.*, 2005)

- **Quality improvement**

  Creating products under the SPL paradigm improves the quality of all products of a product family. The shared components from the platform are reviewed and tested in many products. They have to work properly in more than one kind of product. The extensive quality assurance indicates a significantly higher opportunity of detecting faults and correcting them, thereby improving the quality of all products (Pohl *et al.*, 2005).

- **Reduction of Time-to-market**

  Another very important success factor for a product is the time to market. SPL engineering demands a high upfront investment, which makes time to market initially higher if compared with to single-systems engineering. However, as the reuse of artefacts grow, the time to market is significantly shortened for new products, as can be seen in Figure 2.2.

- **Reduction of Maintenace Effort**

  When a reusable asset from the platform is changed, this change may be propagated to all products in which it is being used. It usually leads to a simpler and cheaper maintenance and evolution, if compared to maintain and evolve a bunch of single products in a separate way.

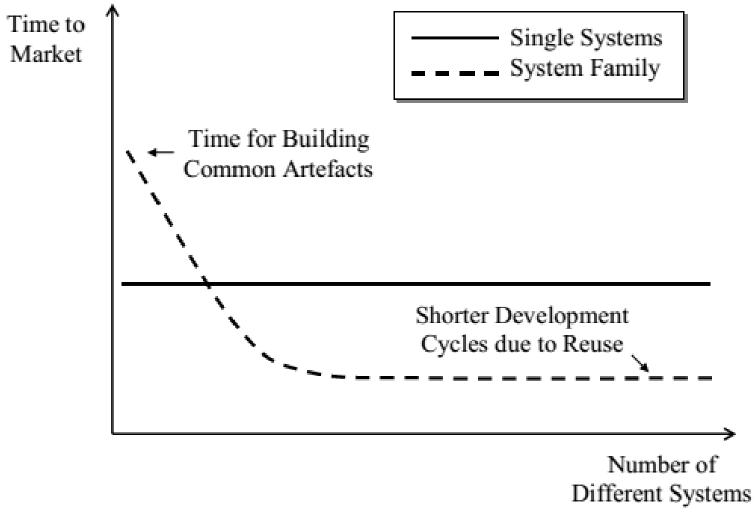

Figure 2.2 Comparison of time to market with and without product line engineering (Pohl *et al.*, 2005)

- **Benefits for the Customers**

  The benefits for the customers are higher quality products at reasonable prices because the production costs become lower in SPL engineering. Besides, products are adapted to their real needs and wishes.

2.1.3 The SPL Development Process

There are a number of different definitions for the Software Product Line (SPL) Development Process on the literature. (Pohl *et al.*, 2005) introduced a framework for SPLE paradigm, shown in Figure 2.3. This framework is divided in two processes:

- **Domain engineering:** This is the process that aims to establish a reusable platform and define the commonality and the variability of the product line. Domain Engineering is composed of five sub-processes: domain requirements, domain design, domain realization, domain testing, and product management (Pohl *et al.*, 2005).

- **Application engineering:** This process is responsible for deriving product line applications from the platform created in domain engineering, where the previously developed components are assembled to compose a product. The application engineering is composed of four sub-processes: application requirements engineering, application design, application realization, and application test (Pohl *et al.*, 2005).

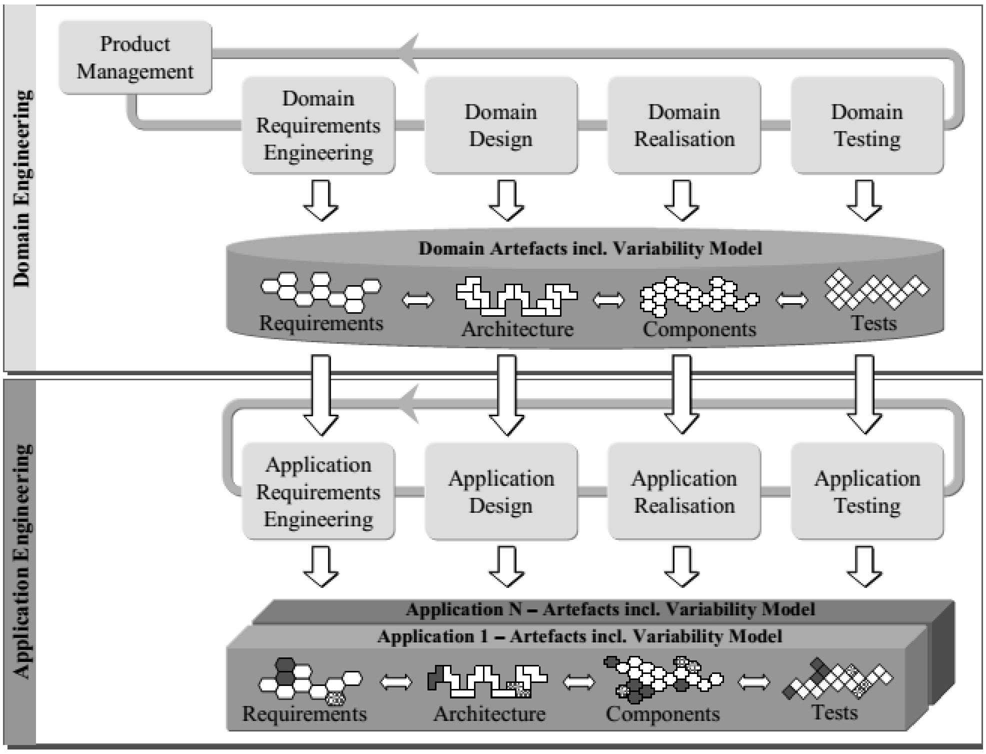

Figure 2.3 The software product line engineering framework (Pohl *et al.*, 2005)

Another popular definition of the Software Product Line (SPL) Development Process can be related to the aforementioned approach. (Clements and Northrop, 2002) defined three essential activities to Software Product Lines: **Core Asset Development (CAD)**, **Product Development (PD)** and **Management activity**, ilustrated in Figure 2.4. In essence, Core Asset Development (CAD) activity is the Domain engineering process, and the Product Development (PD) activity is the Application engineering process. The main difference between these approaches is the Management activity, which is not considered as a process in the first mentioned approach (Pohl *et al.*, 2005).

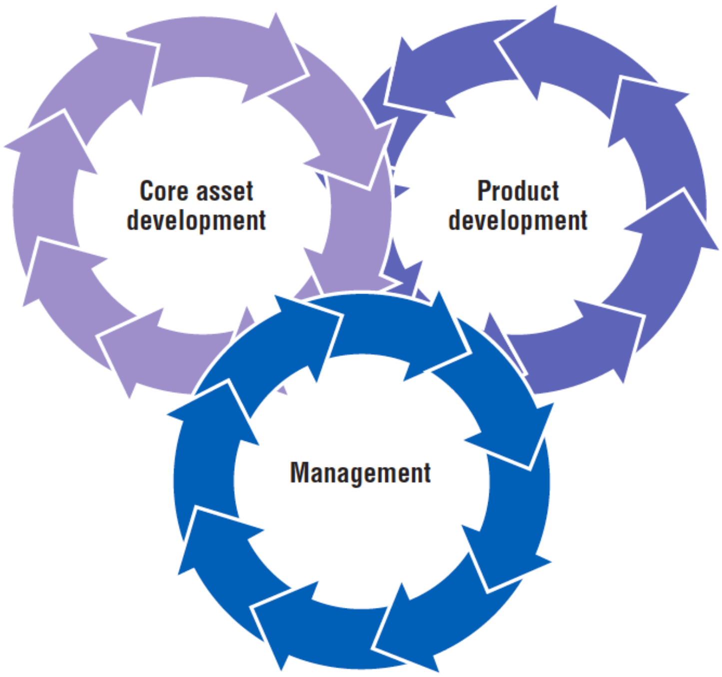

Figure 2.4 SPL Activities (Clements and Northrop, 2002)

Core Asset Development (Domain Engineering)

Core Asset Development (CAD), also called by (Pohl *et al.*, 2005) as domain engineering, is an activity that aims to develop assets to be further reused in other activities. In Figure 2.5, it is shown the core asset development activity, which is interactive, and its inputs and outputs influence each other. The inputs of this activity are product constraints; production constraints; architectural styles; design patterns; application frameworks; production strategy and preexisting assets. This phase is composed of the following sub processes (Pohl *et al.*, 2005):

- **Product Management** deals with the economic aspects associated with the software product line and in particular with the market strategy.

- **Domain Requirements Engineering** involves all activities for eliciting and documenting the common and variable requirements of the product line.

- **Domain Design** encompasses all activities for defining the reference architecture of the product line,

- **Domain Realization** deals with the detailed design and the implementation of reusable software components.

- **Domain Testing** is responsible for the validation and verification of reusable components.

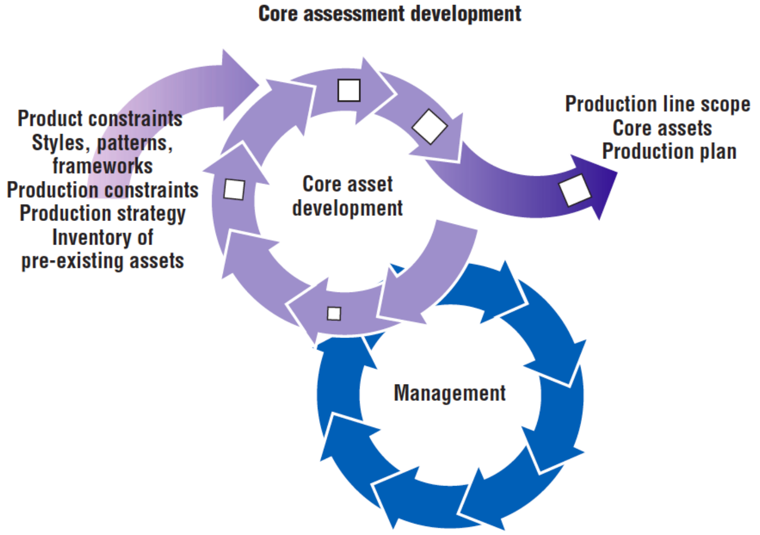

Figure 2.5 Core Asset Development (Clements and Northrop, 2002)

This activity have three outputs: **Product Line Scope**, **Core Assets** and **Production Plan**. The Product Line Scope describes the products that will compose the product line or that the product line can include. This description is recommended to be detailed and well specified, for example, including market analysis activities in order to determine the product portfolio and to encompass which assets and products will be part of the product line. This specification must be driven by economic and business reasons to keep the product line competitive (Capilla *et al.*, 2013).

Core assets are the basis for production of products in the product line. It includes an architecture that will fulfill the needs of the product line, specify the structure of the products and the set of variation points required to support the spectrum of products. It may also include components and their documentation (Clements and Northrop, 2002).

Lastly, the production plan describes how products are produced from the core assets. It details the overall scheme of how the individual attached processes can be fitted together to build a product (Clements and Northrop, 2002). It is what links all the core assets together, guiding the product development within the constraints of the product line.

Product Development (Application Engineering)

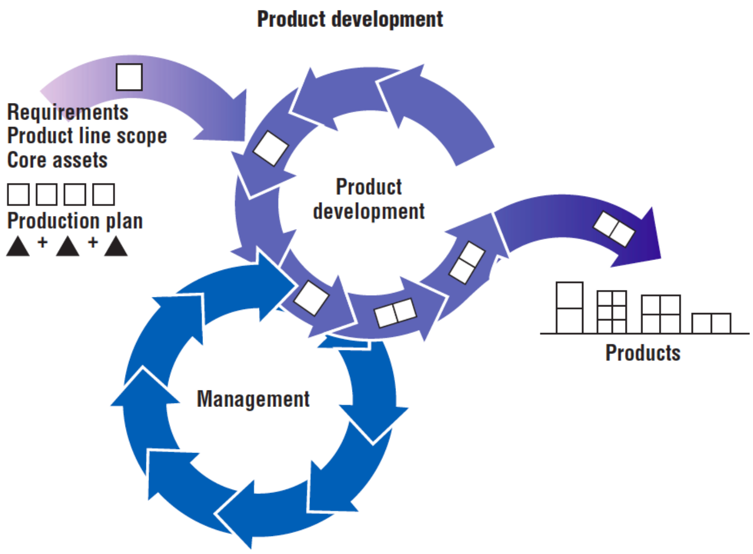

Figure 2.6 Product Development (Clements and Northrop, 2002)

The inputs for this activity are the outputs of the core asset development activity (product line scope, core assets, and production plan) and the requirements specification for individual products as seen in Figure 2.6. The production plan guides how individual products within a product line are constructed using the core assets.

The outputs from this activity should be analyzed by the software engineer and the corrections must be fed back to the Core Asset Development (CAD) activity. During the product development process, some insights happen and it is important to report problems and faults encountered to keep the core asset base healthy.

Management

The management activity is responsible for the production strategy and is vital for success of the product line (Pohl *et al.*, 2005). It is performed in two levels: technical and organizational. The technical management supervise the CAD and PD activities by certifying that both groups that build core assets and products are focused on the activities they are supposed to, and follow the process. The organizational management must ensure that the organizational units receive the right resources in sufficient amounts (Clements and Northrop, 2002).

2.2 Requirements Engineering

Software requirements are descriptions of what the system is expected to do, the services that it must provide and the constraints it must satisfy (Sommerville, 2011). Software requirements are usually classified in a classic way as functional and non-functional. Functional requirements describe what the system must do and non-functional requirements place constraints on how these functional requirements are implemented (Sommerville, 2005).

According to (Sommerville and Kotonya, 1998), Requirements Engineering (RE) is the process by which the software requirements are defined. They state that a process is an organized set of activities that transforms inputs to outputs. Thus, a complete description of a RE process should include what activities are carried out, the structuring or schedule of these activities, who is responsible for each activity and the tools used to support the RE activities.

The RE lifecycle includes requirements elicitation, analysis, negotiation, specification, verification, and management, where (Clements and Northrop, 2002; Sommerville, 2005):

- **Elicitation** identifies sources of requirements information and discovers the users' needs and constraints for the system.
- **Analysis** understands the requirements, their overlaps, and their conflicts.
- **Negotiation** reaches agreement to satisfy all stakeholders, solving conflicts that are identified.
- **Specification** documents the user's needs and constraints clearly and precisely.
- **Verification** checks if the requirements are complete, correct, consistent, and clear.

- **Management** controls the requirements changes that will inevitably arise.

2.3 SPL Requirements Engineering

Requirements are typical assets in SPL. They are specified in reusable models, in which commonalities and variabilities are documented explicitly. Thus, these requirements can be instantiated and adapted to derive the requirements for an individual product (Cheng and Atlee, 2007). During product derivation, for each variant asset, it is decided whether the asset is (or is not) supported by the product to be built. When a domain requirement is instantiated, it can become a concrete product requirement. Thus, new products in the SPL will be much simpler to specify, because the requirements are reused and tailored (Clements and Northrop, 2002).

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

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

- **Elicitation** captures anticipated variations over the foreseeable life-cycle of the SPL. RE must anticipates prospective changes in requirements, such as laws, standards, technology changes, and market needs for future products. Thus, its sources of information are probably larger than for single-system requirements elicitation.

- **Analysis** identifies variations and commonalities, and discovers opportunity for reuse.

- **Negotiation** solves conflicts not only from a logical viewpoint, but also taking into consideration economical and market issues. The SPL requirements may require sophisticated analysis and intense negotiation to agree on both common requirements and variation points that are acceptable for all the systems.

- **Specification** documents a SPL set of requirements. Notations are used to represent the product line variabilities and enable the product instantiation.

- **Verification** checks if the SPL requirements can be instantiated for the products, ensuring the reusability of the requirements.

- **Management** must provide a systematic mechanism for proposing changes, evaluating how the proposed changes will impact the SPL, specifically its core asset base. Evolution can affect the reuse and customization, therefore, appropriate mechanisms must be used o manage the variabilities.

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

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

Several approaches to deal with the definition and specification of functional requirements in SPL development have been proposed over the last few years. Some approaches specify the SPL requirements through features and use cases (Griss *et al.*, 1998; Bayer *et al.*, 2000; Moon *et al.*, 2005; Eriksson *et al.*, 2005; Bonifácio and Borba, 2009; Alférez *et al.*, 2011; Mussbacher *et al.*, 2012; Shaker *et al.*, 2012; de Oliveira *et al.*, 2014). A SPL functional requirement represented as an use case has at least the following fields: identifier, name, description, associated feature(s), pre and post-conditions, and the main success scenario , as shown in Table 2.1. It may also have alternative scenarios, includes/extends relationships, and so on. The feature associated to the use case handles the variability within the SPL.

Table 2.1 SPL Use Case Example (Addapted from (de Oliveira *et al.*, 2014))

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

*Mandatory Fields

However, most of the approaches for specifying SPL functional requirements do not propose guidelines, showing step by step how the specification should be done. This lack of guidelines may lead to some challenges and risks (de Oliveira *et al.*, 2014).

Risks and Challenges

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

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

- **Failure in the communication between core assets requirements development and product requirements development.** The core asset builders need to know the requirements they must build, while the product-specific software builders must know what is expected of them. The lack of communication between these two development stages may lead to inconsistent requirements or even unnecessary variabilities in the requirements.

- **Insufficient generality.** Insufficient generality in the requirements leads to a design that is too fragile to deal with the change actually experienced over the life-cycle of the SPL.

- **Excessive generality.** Excessive generality on requirements leads to excessive effort in producing both core assets (to provide that generality) and specific products (which must turn that generality into a specific instantiation).

- **Wrong variation points.** Incorrect determination of the variation points results in inflexible products and the inability to respond rapidly to customer needs and market shifts.

- **Failure to account for qualities other than behavior.** SPL requirements (and software requirements in general) should capture requirements for quality attributes such as performance, reliability, and security.

2.4 SPLE Tool Support

Since the early days of computer programming, software engineers use a variety of tools to support software development. Software engineering tools and environments are becoming progressively important as the demand for software, its diversity and complexity increases. The computer industry is a competitive industry and there is a pressure to produce software at lower costs and faster because time-to-market is a decisive factor for success. Thus, modern software engineering cannot be accomplished without reasonable tool support (Ossher *et al.*, 2000).

The commercial potential of the SPL approach has already been demonstrated in numerous case studies. While product line development is increasingly accepted, professional tool support is still insufficient and represents a key challenge for future research (Pohl *et al.*, 2005; Schmid *et al.*, 2006).

SPLE tool support focuses almost exclusively on a single, cross-cutting aspect of SPLE: variability management Variability Management (VM), or making software and artifacts (such as requirements, tests, and documentation) configurable in a way that they can be developed together, while each product still receives its specifically adapted version (Schmid and Santana de Almeida, 2013). Thus, an effective and efficient variability management VM is the base of the successful reuse of development artifacts (Boutkova, 2011).

Variability Management (VM) tools support four main activities: modeling variability, modeling the relationship between variability and a generic artifact, supporting configuration of generic artifacts, and deriving customized products (Schmid and Santana de Almeida, 2013).

The requirements engineering process must be tool-supported to handle the huge volume of elicited requirements. There are several differences between a single product development and a product line development and therefore a tool must be capable to support that development, including the additional activities that must be performed in the requirements engineering phase. However, existing tools are not designed to support the requirements engineering process for software product lines. Existing tools support only single product development and therefore lack support for modeling commonalities and variabilities as well as variation points in requirements (Birk *et al.*, 2003).

2.5 Summary

In this chapter, we discussed about important concepts to this work: the area of Software Product Line (SPL), Requirements Engineering (RE) , SPL Requirements Engineering and SPLE tool support.

Next chapter presents an extension of Software Product Line Integrated Construction Environment (SPLICE), a web-based, collaborative support for the SPL lifecycle steps.

Bibliography

Alférez, M., Lopez-Herrejon, R. E., Moreira, A., Amaral, V., and Egyed, A. (2011). Supporting consistency checking between features and software product line use scenarios. In *Top Productivity through Software Reuse*, pages 20–35. Springer.

Almeida, E. S., Alvaro, A., Lucrédio, D., Garcia, V. C., and Meira, S. R. L. (2004). Rise project: Towards a robust framework for software reuse. In *IEEE International Conference on Information Reuse and Integration (IRI)*, pages 48–53, Las Vegas, NV, USA.

Alvaro, A., Almeida, E. S., and Meira, S. L. (2006). A software component quality model: A preliminary evaluation. In *Proceedings of the 32nd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO'06)*, pages 28–37, Washington, DC, USA. IEEE Computer Society.

Bayer, J., Muthig, D., and Widen, T. (2000). Customizable domain analysis. In *Generative and Component-Based Software Engineering*, pages 178–194. Springer.

Birk, A., Heller, G., John, I., Joos, S., Muller, K., Schmid, K., and von der Massen, T. (2003). Report of the gi work group" requirements engineering for product lines.

Bonifácio, R. and Borba, P. (2009). Modeling scenario variability as crosscutting mechanisms. In *Proceedings of the 8th ACM international conference on Aspect-oriented software development*, pages 125–136. ACM.

Boutkova, E. (2011). Experience with variability management in requirement specifications. In *Software Product Line Conference (SPLC), 2011 15th International*, pages 303–312. IEEE.

Brito, K. S. (2007). *LIFT: A Legacy InFormation retrieval Tool*. Master's thesis, Federal University of Pernambuco, Recife, Pernambuco, Brazil.

Capilla, R., Bosch, J., and , K. (2013). *Systems and Software Variability Management: Concepts, Tools and Experiences*. SpringerLink : Bücher. Springer.

Cavalcanti, R. d. O., de Almeida, E. S., and Meira, S. R. (2011). Extending the riple-de process with quality attribute variability realization. In *Proceedings of the Joint ACM SIGSOFT Conference – QoSA and ACM SIGSOFT Symposium – ISARCS on Quality of Software Architectures – QoSA and Architecting Critical Systems – ISARCS*, QoSA-ISARCS '11, pages 159–164, New York, NY, USA. ACM.

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.

Cavalcanti, Y. C., do Carmo Machado, I., da Mota Silveira Neto, P. A., and Lobato, L. L. (2012). *Software Product Line - Advanced Topic*, chapter Handling Variability and Traceability over SPL Disciplines, pages 3–22. InTech.

Chastek, G., Donohoe, P., Kang, K. C., and Thiel, S. (2001). Product line analysis: a practical introduction. Technical report, DTIC Document.

Cheng, B. H. and Atlee, J. M. (2007). Research directions in requirements engineering. In *2007 Future of Software Engineering*, pages 285–303. IEEE Computer Society.

Clements, P. and Northrop, L. (2002). *Software Product Lines: Practices and Patterns*. The SEI series in software engineering. Addison Wesley Professional.

da Mota Silveira Neto, P. A. (2010). A regression testing approach for software product lines architectures.

de Oliveira, R. P., Blanes, D., Gonzalez-Huerta, J., Insfran, E., Abrahão, S., Cohen, S., and de Almeida, E. S. (2014). Defining and validating a feature-driven requirements engineering approach. *Journal of Universal Computer Science*, **20**(5), 666–691.

de Oliveira, T. H. B. (2009). Riple-em: A process to manage evolution in software product lines.

de Souza Filho, E. D. (2010). Riple-de: A domain design process for software product lines.

do Carmo Machado, I. (2010). Riple-te: A software product lines testing process.

Durao, F. A. (2008). *Semantic Layer Applied to a Source Code Search Engine*. Master's thesis, Federal University of Pernambuco, Recife, Pernambuco, Brazil.

Eriksson, M., Börstler, J., and Borg, K. (2005). The pluss approach–domain modeling with features, use cases and use case realizations. In *Software Product Lines*, pages 33–44. Springer.

Garcia, V. C., Lisboa, L. B., ao, F. A. D., Almeida, E. S., and Meira, S. R. L. (2008). A lightweight technology change management approach to facilitating reuse adoption.

In *2nd Brazilian Symposium on Software Components, Architectures, and Reuse (SBCARS'08)*, Porto Alegre, Brazil.

Griss, M. L., Favaro, J., and Alessandro, M. D. (1998). Integrating feature modeling with the rseb. In *Software Reuse, 1998. Proceedings. Fifth International Conference on*, pages 76–85. IEEE.

Lisboa, L. B. (2008). Toolday - a tool for domain analysis.

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 (SBCARS'08)*, Porto Alegre, Brazil.

Mascena, J. C. C. P. (2006). *ADMIRE: Asset Development Metric-based Integrated Reuse Environment*. Master's thesis, Federal Uniersity of Pernambuco, Recife, Pernambuco, Brazil.

Melo, Buregio, E. S. A. M. (2008). A reuse repository system: The core system. In *Proc. of ICSR*.

Mendes, R. C. (2008). *Search and Retrieval of Reusable Source Code using Faceted Classification Approach*. Master's thesis, Federal University of Pernambuco, Recife, Pernambuco, Brazil.

Moon, M., Yeom, K., and Chae, H. S. (2005). An approach to developing domain requirements as a core asset based on commonality and variability analysis in a product line. *Software Engineering, IEEE Transactions on*, **31**(7), 551–569.

Moraes, M. B. S. (2010). *A scoping approach for software product lines*. Master's thesis, Federal University of Pernambuco, Recife, Pernambuco, Brazil.

Mussbacher, G., Araújo, J., Moreira, A., and Amyot, D. (2012). Aourn-based modeling and analysis of software product lines. *Software Quality Journal*, **20**(3-4), 645–687.

Nascimento, L. M. (2008). *Core Assets Development in SPL - Towards a Practical Approach for the Mobile Game Domain*. Master's thesis, Federal University of Pernambuco, Recife, Pernambuco, Brazil.

Neiva, D. F. S. (2009). Riple-re: A requirements engineering process for software product lines.

Ossher, H., Harrison, W., and Tarr, P. (2000). Software engineering tools and environments: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering*, pages 261–277. ACM.

Pohl, K., Böckle, G., and van der Linden, F. (2005). *Software Product Line Engineering: Foundations, Principles, and Techniques*.

Ribeiro, H. B. G. (2010). An approach to implement core assets in service-oriented product lines.

Santos, E. C. R., ao, F. A. D., Martins, A. C., Mendes, R., Melo, C., Garcia, V. C., Almeida, E. S., and Meira, S. R. L. (2006). Towards an effective context-aware proactive asset search and retrieval tool. In *6th Workshop on Component-Based Development (WDBC'06)*, pages 105–112, Recife, Pernambuco, Brazil.

Schmid, K. and Santana de Almeida, E. (2013). Product line engineering. *Software, IEEE*, **30**(4), 24–30.

Schmid, K., Krennrich, K., and Eisenbarth, M. (2006). Requirements management for product lines: extending professional tools. In *Software Product Line Conference, 2006 10th International*, pages 10–pp. IEEE.

Shaker, P., Atlee, J. M., and Wang, S. (2012). A feature-oriented requirements modelling language. In *Requirements Engineering Conference (RE), 2012 20th IEEE International*, pages 151–160. IEEE.

Sommerville, I. (2005). Integrated requirements engineering: A tutorial. *Software, IEEE*, **22**(1), 16–23.

Sommerville, I. (2011). *Software Engineering*. Addison Wesley, 9 edition.

Sommerville, I. and Kotonya, G. (1998). *Requirements engineering: processes and techniques*. John Wiley & Sons, Inc.

Thurimella, A. K. and Bruegge, B. (2007). Evolution in product line requirements engineering: A rationale management approach. In *Requirements Engineering Conference, 2007. RE'07. 15th IEEE International*, pages 254–257. IEEE.

Vanderlei, T. A., ao, F. A. D., Martins, A. C., Garcia, V. C., Almeida, E. S., and Meira, S. R. L. (2007). A cooperative classification mechanism for search and retrieval software components. In *Proceedings of the 2007 ACM symposium on Applied computing (SAC'07)*, pages 866–871, New York, NY, USA. ACM.