Extractive Product Line Requirements Engineering

by

Nan Niu

A thesis submitted in conformity with the requirements for the degree of Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto

Copyright © 2009 by Nan Niu
Abstract

Extractive Product Line Requirements Engineering

Nan Niu
Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto
2009

A software product line (SPL) succeeds because we exploit the similarities between a set of software-intensive systems, together with an understanding of their differences, to reduce development cost, maintenance cost, and user confusion. In SPL engineering, reuse is planned, enabled, and enforced. It is through the development of a set of core assets that reuse is systematically practiced. Requirements assets enhance the effectiveness of reuse since engineers can work on the abstractions closer to the systems’ initial concepts.

Contemporary SPL requirements engineering (RE) approaches often adopt the proactive model to build a relatively complete and stable asset base. In practice, the substantial up-front effort and the abrupt transition from existing practices associated with the proactive model present a prohibitive SPL adoption barrier for many organizations that could otherwise benefit. The extractive model overcomes these shortcomings by reusing existing products for the SPL’s initial baseline.

In this thesis, we present a framework for applying lightweight techniques to extract, model, and analyze a SPL’s requirements assets. We define the notion of functional requirements profiles (FRPs) according to the linguistic characterization of a domain’s action-oriented concerns, and show that the FRPs can be extracted from a natural language document on the basis of domain-aware lexical affinities that bear a ‘verb - direct object’ relation. We model the extracted FRPs by analyzing their semantic cases and by extending the orthogonal variability model (OVM). We contribute a set of heuristic
rules for uncovering the variation dimensions and dependencies, and discuss merging the
OVMs extracted from multiple sources. We relate functional profiles to quality require-
ments via scenarios, and manage requirements interactions via concept analysis. We
present two applications of FRPs to support some other activities in SPL engineering.

We conduct several empirical studies to evaluate our framework. The results show
that our approach allows the engineers to identify the domain elements more easily and
develop the domain models more systematically. Our work fills the void with respect to
extracting a SPL’s requirements assets, and the main thrust of our work is to promote a
set of lightweight, low adoption threshold techniques as a critical enabler for practitioners
to capitalize on the order-of-magnitude improvements offered by SPL engineering.
To Jia, my love.
Acknowledgements

First and foremost, I would like to express my sincere gratitude to my supervisor, Steve Easterbrook, for accepting me and supporting me throughout my studies. He has lent invaluable knowledge, encouragement, and guidance to my work. I thank him for the many hours he has spent with me, exploring ideas, examining the details, challenging my thoughts, improving my skills, and discussing with me everything from research and career to family and life. He has showed great patience and provided me the degree of freedom necessary for a creative atmosphere. His gentleness, earnestness, and positiveness have always impressed me, and will continue to inspire me in my future career.

My cordial thanks also go to my supervisory and defense committees: John Mylopoulos, Eric Yu, Arno Jacobsen, and Krzysztof Czarnecki. They have taught me much, and their advices have made my Ph.D. experience both more educational and more enjoyable. While at UoT, I had the privilege of interacting with wonderful and talented Yijun Yu, Sotirios Liaskos, Mehrdad Sabetzadeh, Shiva Nejati, Jorge Aranda, Lin Mei, Wendy Liu, Jennifer Horkoff, Neil Ernst, Ou Wei, Yuntian (Jean) Fan, Yinhua (Sylvia) Jia, and Haifeng (Lisa) Liu. It has been a real pleasure to work with you all.

I thank Professors Klaus Pohl, Tom Maibaum, Daniel Berry, Anthony Finkelstein, Colette Rolland, and Jennifer Campbell for directing my research in various stages. I am very grateful to Mara Bullock and Geoff Knapp for coordinating the case study with IBI. I also want to thank my friends, most notably, my fellow students from the Math Department and my former Lenovo colleagues who built their new homes in Toronto. Thanks for all the fun and happiness you brought to me.

Finally, I am so blessed to have a wonderful family that unconditionally supports my pursuing my goals: my parents, my parents-in-law, and my angel, Victoria, who brings so much joy to the family. Most of all, I dedicate the thesis to my beloved wife, Jia Wang. Without her relentless support, I would not have been so focused and firm in pursuing my Ph.D. — Yes, honey, I did it, finally!
## Contents

1 Introduction 1
   1.1 Motivation 1
   1.2 Scope 4
   1.3 Contributions 8
   1.4 Organization 10

2 Background 12
   2.1 The Role of RE in SPL Engineering 13
      2.1.1 SPL Philosophy 13
      2.1.2 Requirements as Core Assets 15
      2.1.3 RE and Domain Analysis 17
   2.2 Plan and Elicit 20
      2.2.1 Two-Tiered Organization 20
      2.2.2 Scoping 22
      2.2.3 Elicitation 25
   2.3 Model and Analyze 30
      2.3.1 Goal Modeling 30
      2.3.2 Use Case Modeling 33
      2.3.3 Feature Modeling 36
      2.3.4 Propositional Formulas 39
# List of Tables

1.1 Mapping thesis chapters to key publications ........................................... 11

2.1 Software product line scoping ................................................................. 23

2.2 Elicitation techniques .............................................................................. 25

2.3 A PR-context matrix .................................................................................. 28

2.4 Summary of SPL requirements approaches ........................................... 52

3.1 Profiling the AMS SRS (3,264 words) ...................................................... 62

4.1 Top 5 FRPs extracted ................................................................................. 98

4.2 Measuring product similarity based on FRPs ........................................ 99

5.1 Extracted requirements constructs ............................................................ 114

5.2 Crosscutting relations .............................................................................. 115

5.3 Extractive analysis results ......................................................................... 132

6.1 FRP-attribute matrix (cf. Figure 6.1b) ...................................................... 144

6.2 Dissimilarity indices ($\delta$) of Table 6.1 .................................................. 145

6.3 Similarity levels ($\sigma$) of Table 6.2 .......................................................... 146

6.4 Normalized matrix of Table 6.1 ................................................................. 150

6.5 Information loss ($\delta I$) of Table 6.4 ......................................................... 151
# List of Figures

1.1 Overview of the activities ........................................ 7

2.1 SADT context view of domain analysis ............................ 18

2.2 Domain engineering and application engineering processes .......... 19

2.3 Essential product line activities ..................................... 21

2.4 A goal model ......................................................... 31

2.5 A domain use case diagram ........................................... 34

2.6 A use case diagram including variation points and variants ............ 35

2.7 Node-based and edge-based semantics .................................. 37

2.8 A feature diagram and its grammar .................................... 40

2.9 GUI specification .................................................... 40

2.10 Prioritization in a definition hierarchy ............................... 47

3.1 FRPs extraction example .............................................. 56

3.2 Extracting domain-aware LAs ......................................... 60

3.3 Evaluation procedure for the auto-marker study ....................... 65

3.4 Comparing FRPs with single-term indices ............................ 68

3.5 Determining threshold ................................................ 70

3.6 Evaluation procedure for the IBI study ................................ 74

3.7 Extraction effectiveness comparison .................................... 76

3.8 Improvement made by filtering out document-formatting stop words ... 78
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.9</td>
<td>Investigating recalls</td>
<td>79</td>
</tr>
<tr>
<td>4.1</td>
<td>A sample OVM</td>
<td>84</td>
</tr>
<tr>
<td>4.2</td>
<td>Relating an OVM to a use case model</td>
<td>84</td>
</tr>
<tr>
<td>4.3</td>
<td>Semantic cases and OVM</td>
<td>88</td>
</tr>
<tr>
<td>4.4</td>
<td>Partial OVM for the auto-marker SPL</td>
<td>90</td>
</tr>
<tr>
<td>4.5</td>
<td>OVM for “calculate travel time”</td>
<td>95</td>
</tr>
<tr>
<td>4.6</td>
<td>OVM for “detect queue”</td>
<td>100</td>
</tr>
<tr>
<td>4.7</td>
<td>OVM for “generate response plan”</td>
<td>101</td>
</tr>
<tr>
<td>5.1</td>
<td>Binary relation</td>
<td>109</td>
</tr>
<tr>
<td>5.2</td>
<td>Concept lattice in sparse representation</td>
<td>111</td>
</tr>
<tr>
<td>5.3</td>
<td>Concept lattice for the running example</td>
<td>117</td>
</tr>
<tr>
<td>5.4</td>
<td>Categorizing the concept lattice</td>
<td>119</td>
</tr>
<tr>
<td>5.5</td>
<td>Detecting interferences</td>
<td>121</td>
</tr>
<tr>
<td>5.6</td>
<td>Updating concept hierarchy incrementally</td>
<td>125</td>
</tr>
<tr>
<td>5.7</td>
<td>Analyzing trade-offs on a sliced lattice</td>
<td>129</td>
</tr>
<tr>
<td>5.8</td>
<td>Products in the Genuine Soccer SPL</td>
<td>131</td>
</tr>
<tr>
<td>6.1</td>
<td>Running example (library MIS)</td>
<td>141</td>
</tr>
<tr>
<td>6.2</td>
<td>Semantic attributes (library MIS)</td>
<td>143</td>
</tr>
<tr>
<td>6.3</td>
<td>Overlapping clusters of Table 6.3</td>
<td>148</td>
</tr>
<tr>
<td>6.4</td>
<td>Dendrogram of hierarchical clusters</td>
<td>153</td>
</tr>
<tr>
<td>6.5</td>
<td>Part of overlapping clusters</td>
<td>154</td>
</tr>
<tr>
<td>6.6</td>
<td>Domain feature diagram</td>
<td>156</td>
</tr>
<tr>
<td>6.7</td>
<td>Terminology and concepts</td>
<td>162</td>
</tr>
<tr>
<td>6.8</td>
<td>A sample repertory grid</td>
<td>165</td>
</tr>
<tr>
<td>6.9</td>
<td>List of all elements and constructs</td>
<td>171</td>
</tr>
<tr>
<td>6.10</td>
<td>A projection of the resultant grid</td>
<td>172</td>
</tr>
</tbody>
</table>
6.11 Aligning auto-marker NFRs .................................................. 175
6.12 Some naming conventions proposed for $i^*$ modeling framework ........ 179
Chapter 1

Introduction

1.1 Motivation

Product lines are nothing new in manufacturing. Airbus builds one, and so do Ford, Dell, and even McDonald’s. Software product line (SPL) engineering is one of the success stories of software reuse whose purpose is to improve software quality and productivity [51]. A SPL succeeds because the commonalities shared by the software-intensive systems can be exploited to achieve economies of production [35, 125].

A SPL is defined as “a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way” [35]. A key concept is the set of core assets, which include those artifacts in software development that are most costly to develop from scratch – namely, the requirements, domain models, architecture, test cases, and components [125]. The core assets are designed for reuse, such that individual systems can be designed with reuse. Therefore, building and evolving the core assets are crucial and beneficial in SPL engineering. This very benefit, dialectically speaking, can become a burden in that it is not usually worth building the set of core assets by itself — this set is not a product that anyone would ask for [121].
Although much of the SPL research to date has focused on reusing architectural design and code, anecdotal evidence abounds in support of treating requirements as an asset. Core, a SPL for avionics simulators, owed its success largely to the conceptual analysis of requirements and the development of a generic requirements specification [9]. Not only were reuse opportunities identified early in the software life cycle, but also the effectiveness of reuse was enhanced as developers can work on the abstractions closer to the systems’ initial concepts [82].

Many contemporary SPL methods, such as FODA (feature-oriented domain analysis) [75] and FAST (family-oriented abstraction, specification, and translation) [161], base requirements definition on domain analysis [128], and so do many requirements engineering (RE) techniques for SPLs like PRS (product-line requirements specification) [48] and definition hierarchies [85]. One of the drawbacks of domain analysis refers to its intrinsic domain dependence: it counts on experts’ experience and intuition to manually acquire domain requirements. Namely, there are no rules that enable engineers to identify domain elements easily [100]. This makes domain analysis a labor-intensive and error-prone process, and presents a prohibitive SPL adoption barrier for many organizations that could otherwise benefit. In another word, if “design for reuse” is too costly, nobody could afford to “design with reuse”.

Current approaches often carry out a SPL’s domain analysis in a proactive manner. The premise is that building a relatively stable and more complete domain model pays off in subsequent application engineering. Being proactive requires substantial up-front effort, yet it is difficult to predict the return on investment. Proactive SPL adoption dictates a significant change in an organization’s existing practice and demands compromise from software practitioners [130]. For example, for Cummins to achieve its impressive SPL successes, it stopped all product deployments for six months while it rearchitected its engine control software, organizational charts, and processes [34].

Parnas aptly summarized the dilemma faced in greenfield SPL development: we had
to design the core assets for a product family at a time when we could not possibly know what members of the family would actually be built [122]. To resolve such a paradox and to ease the transition from a single-system mentality to software mass customization, Krueger proposed the extractive adoption model as a means of reusing existing products for the SPL’s initial baseline [83]. Core assets are no longer created from scratch, but are mined from software repositories. The extractive approach is particularly effective for an organization that has accumulated development experiences and artifacts in a domain and wants to quickly transition from conventional to SPL engineering. Notably, the main beneficiaries are small and medium-sized enterprises (SMEs) as large companies tend to proactively launch a SPL in the mature market segment [22]. The basic tenets of the extractive model are: 1) Mining legacy software repositories readily spots reuse opportunities; 2) Incrementally exposing small variations avoids over-specifying and inaccurately predicting a complete feature set; and 3) Under-specified assets will be enriched when abstractions are refactored as they emerge from an evolving SPL.

In [23], the authors reported one of the first case studies of the extractive SPL adoption model. Compared to the effort typically reported with proactive SPL transition approaches, the company achieved two orders-of-magnitude savings (measured by engineer-months) by adopting a combination of extractive and reactive approaches. The study focused on extracting and evolving the SPL’s architecture, design, and source code modules. In fact, the literature has paid little attention to extracting a SPL’s requirements assets.

Our goal is to complement existing methods by leveraging the extractive model in engineering a SPL’s requirements. We aim at filling the extraction void with lightweight techniques. By lightweight, we indicate that the methods employed are striving for increased automation and greater generality, and that the methods can be used to perform partial analysis without a commitment to developing a complete specification of the domain. When full automation is impractical, we want to codify a set of rules, heuristics,
and guidelines that help engineers identify the domain elements more easily and develop the domain model more effectively.

1.2 Scope

Requirements engineering (RE) is a set of activities concerned with identifying and communicating the purpose of a software-intensive system, and the contexts in which it will be used. Core RE activities include: plan and elicit requirements, model and analyze requirements, communicate and agree requirements, and realize and evolve requirements [115]. In addition, managing commonality and variability is a central theme running throughout the RE activities for a SPL [37].

In this thesis, we focus on extracting, modeling, and analyzing a SPL’s requirements assets. We tackle such fundamental problems as where and what to extract, how to represent the results, and how to use the results to support other activities in SPL engineering. In the following, we introduce the background of our work, highlight the specific problems that are investigated in this thesis, and outline our solutions to these problems. The discussion is organized by the three activities that we concentrate on: extracting, modeling, and analyzing.

Extracting. The problems that we address in this activity are where, what, and how to extract requirements assets. Studies of RE practice in SMEs, such as [53] and [4], showed that the majority of requirements are written in natural language (NL) because text is used universally to convey information and to communicate. We therefore choose NL documents to be the primary extraction source, and anticipate the textual-based technique can effect a wide spectrum of domains.

When constructing a SPL’s requirements assets, we shall follow two principles [125]:
1) Focus more on external variability (visible to customers) and less on internal variability
(useful to implementers), 2) Focus more on what varies (variation point) and less on how it varies (variants). Our strategy is to tease out what we call the functional requirements profiles (FRPs) from the NL documents [108]. The functional requirements represent such salient features directly observable by the users and customers [76] that an external view is obtained.

We define the notion of FRP to capture the domain’s action themes at a primitive level, and to define a context for studying system qualities [20]. The FRPs in each document are identified on the basis of lexical affinities [93] and “verb–direct object” relations [147]. We have developed an information retrieval (IR) technique to automatically extract the FRPs from the textual requirements [108]. We prefer IR techniques in our work for reasons of cost, scalability, and domain transportability [93].

**Modeling.** In this activity, we are concerned mainly with representing the extraction results in such a way that the commonality and variability properties can be better understood and the stakeholders to whom the FRPs are relevant are represented more explicitly. Another key issue is to match and merge the extracted requirements constructs from different sources in order to form the SPL’s initial asset base.

We adopt one of the most recently developed SPL modeling framework, the orthogonal variability model (OVM) [125], to represent the extraction results. An OVM defines a SPL’s variability in a single view, so we can consistently manage the variability across requirements, design, realization, and testing artifacts. The building blocks of the OVM are variation points, variants, dependencies, and constraints. We treat the validated FRPs as primary constructs when establishing the OVM since FRPs capture the domain’s action-oriented concerns and every product in the SPL should address these concerns in one way or another. We use Fillmore’s case theory [50] and draw information from recent work on variability frames for goals [90] to characterize each FRP’s semantics, and further propose some heuristics for uncovering the variation dimensions and dependencies [108]. This helps to build an OVM from the FRPs extracted from a single document.
When dealing with a set of requirements documents, we want to compare, contrast, and integrate the FRPs, and the corresponding OVMs, extracted from different sources. We view this problem as an instance of model matching and merging [104]. Our matcher takes advantage of the domain-specific thesaurus and the domain-neutral WordNet [164] to establish correspondences among the model elements. Our merger [21] then uses the identified relationships and acts as conjunction [171] in forming the SPL’s asset base. We also discuss how to handle variability in the merge processes.

**Analyzing.** We describe three kinds of analysis of the extracted requirements assets to support some other activities in SPL engineering. First of all, to account for the quality requirements \(^1\), such as reliability and usability, we make use of the SEI’s quality attribute scenarios [10] and perform formal concept analysis for the SPL’s requirements [110]. The concept lattice resulted from the analysis provides a rich notion that allows remarkable insights into the modularity and interactions of requirements. By manipulating the concept lattice, we address such issues as locating quality-specific functional units, detect interferences, update the concept lattice incrementally, and analyze the change impact.

Secondly, analyzing the semantic cases allows us to cluster the FRPs [109]. Clustering fits our framework’s lightweight philosophy because it is an example of unsupervised learning, meaning it does not rely on predefined classes or manually-labeled training set [60]. Our clustering algorithms attempt to organize and find structure in requirements. We recognize stakeholders’ different goals when clustering a SPL’s requirements, and introduce an on-demand framework to accommodate these goals [109]. For example, from the user’s viewpoint, clustering helps identify, browse, and prioritize features; from the designer’s viewpoint, clustering helps achieve system decomposition and modularization.

Our third analysis uses ideas from psychology to identify terminological interferences

---

\(^1\)We have chosen to use the umbrella term “quality requirements” [15] in this thesis to mean requirements that describe desired system qualities, which are also known as nonfunctional requirements, softgoals, and quality attributes in the literature [115]. This chosen terminology shall not be confused with high-quality requirements.
when different stakeholders state their requirements or build requirements models [107]. The key is to use concrete entities, such as FRPs or tasks, to establish a common ground among the stakeholders, and to compare stakeholders’ constructs by how they relate to the set of concrete entities rather than by any terms the stakeholders use to describe them. Our analysis shows that, in extractive SPL RE, the primitives better reveal the context.

Figure 1.1 shows an overview of the main activities discussed in this thesis, and also specifies the chapters associated with these activities. The central construct is the notion of FRPs, and the fundamental activity is to extract the FRPs from a requirements document (Chapter 3). Chapter 4 is concerned with modeling the FRPs, and the major outputs are FRP-based OVMs. We divide the analysis into two parts: Chapter 5 accounts for NFRs and Chapter 6 shows two applications of FRPs in SPL engineering. We devote a single chapter to NFRs because addressing NFRs should be an integral part of any RE framework. We describe two applications of FRPs in Chapter 6: cluster analysis and detecting terminological interferences. Both applications take advantage of the FRPs’ primitive and homogeneous level of abstraction.

The activities shown in Figure 1.1 form our framework for applying lightweight tech-
niques to do RE for a SPL in an extractive manner. Note that one may extract other constructs than FRPs, choose other notations than OVMs for modeling, or employ other methods to analyze the assets. The scope of this thesis is to present a coherent set of techniques that center around the construct of FRPs.

1.3 Contributions

The theme of our work is the study of how extractive techniques can improve the state-of-the-art practices in SPL RE. We not only propose a framework for applying lightweight techniques to extract, model, and analyze a SPL’s requirements assets (cf. Figure 1.1), but also carry out a number of empirical studies to assess the applicability and usefulness of our methods. The framework is one of the first attempts to deal with doing RE for a SPL in an extractive manner. This thesis provides the following contributions:

- The definition of the functional requirements profile (FRP), a construct based on linguistic clues that is capable of modeling system function and action theme of the application domain.

- Algorithms for automatically extracting FRPs from natural language documents using a combination of information retrieval (IR) and natural language processing (NLP) techniques.

- The semantic analysis, based on Fillmore’s case theory [50], for uncovering the FRP’s variation points (what dimensions the FRP can vary) and variants (how the FRP varies along a particular dimension).

- A set of heuristics for determining the intra-FRP variability (i.e., the mandatory or optional property of the variant) and the inter-FRP variability (i.e., the mutual dependence and exclusion constraints).
• The extension of the orthogonal variability model (OVM) [125] that includes the FRPs as primitives to model functional requirements, and that includes visualization aids to facilitate comprehension and communication.

• The comparison of products based on FRP-differences, and a linguistic-based procedure for matching and merging the FRPs and the OVMs.

• The use of quality attribute scenarios [10] to relate FRPs to quality requirements, and to further generate a concept lattice that allows remarkable insights into the modularity and interactions of requirements.

• The novel application of formal concept analysis [54] that addresses the following issues by manipulating the concept lattice:
  – Locating quality-specific functional units;
  – Detecting interferences;
  – Updating the concept hierarchy incrementally; and
  – Analyzing the impact of changing requirements as the SPL evolves.

• The management of terminological interferences of quality requirements and of SPL features by using the ideas from psychology [78] and by using the FRPs to establish the context.

• The introduction of an on-demand clustering framework that recognizes stakeholders’ different goals in analyzing the SPL’s requirements.

• Advancing the requirements clustering literature by examining clusters that overlap and those causing a minimal information loss.

• The design and implementation of the proposed algorithms, and a number of empirical studies that evaluate our approach, including the analysis of an auto-marker SPL, a family of traffic control systems, and a mobile game SPL.
The overall contribution of the thesis is the development of techniques to support extractive RE for SPLs and the demonstration of the benefits of the proposed methods in building reusable assets. The main thrust of our work is to promote a set of lightweight, low adoption threshold techniques as a critical enabler, which complements existing RE approaches for SPLs, for practitioners to capitalize on the order-of-magnitude improvements offered by SPL engineering.

1.4 Organization

The remainder of this thesis is organized as follows:

Background

Chapter 2 presents necessary background on the role of RE in SPL engineering, and presents an overview of the state-of-the-art in modeling the SPL’s requirements assets, including goal modeling, use case modeling, and feature modeling.

Extracting Functional Requirements Profiles

Chapter 3 details the rationale behind our extraction of the functional requirements profiles (FRPs), a concrete definition of the FRP, and an FRP extraction algorithm based on information retrieval and natural language processing techniques. Two evaluations are presented: one on an auto-marker SPL and the other on a family of traffic control systems.

Modeling Requirements Assets

Chapter 4 describes the semantic analysis of the FRPs. We further use the obtained semantic information to identify the FRP’s variation dimensions, and both intra-FRP and inter-FRP variability constraints so as to build an extended orthogonal variability model. We then present the matching and merging of the requirements models extracted from different sources. We continue to study the auto-marker and traffic control SPLs with respect to the modeling aspect.
Chapter 1. Introduction

Table 1.1: Mapping thesis chapters to key publications

<table>
<thead>
<tr>
<th>Chapter 3</th>
<th>RE’08 [108]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter 4</td>
<td>RE’08 [108]</td>
</tr>
<tr>
<td>Chapter 5</td>
<td>AOSD’09 [110]</td>
</tr>
<tr>
<td>Chapter 6</td>
<td>Section 6.1 SPLC’08 [109]</td>
</tr>
<tr>
<td></td>
<td>Section 6.2 IEEE Software [107]</td>
</tr>
</tbody>
</table>

Analyzing Quality Requirements

Chapter 5 presents our treatment of quality requirements: the use of quality attribute scenarios to relate functional profiles to quality requirements, and the concept analysis of both functional and quality requirements in the presence of an evolving SPL. We use a mobile game SPL for evaluation.

Applications of FRPs in Product Line Engineering

Chapter 6 presents two applications of the extracted FRPs: on-demand clustering that takes stakeholders’ different goals into account, and managing interferences when stakeholders do not use their terminologies consistently. A library management information system and a social service organization’s requirements project are used for demonstration.

Conclusions and Future Work

Finally, we conclude in Chapter 7 with a summary of the thesis and our future research directions.

Table 1.1 shows a mapping between thesis chapters and our key publications. The mapping is only indicative in that modifications of the published work are made to fit the thesis’s theme, and additional work has been added to the thesis, most notably, the IBI case study reported in Chapter 3 and Chapter 4.
Chapter 2

Background

A SPL succeeds because we exploit the similarities between the products, together with an understanding of the differences, to reduce development costs, maintenance costs, and user confusion. This idea was first suggested in designing and developing program families:

“Program families are defined (analogously to hardware families) as sets of programs whose common properties are so extensive that it is advantageous to study the common properties of the programs before analyzing individual members.” [120]

The common properties and permissible variabilities are captured as core assets in the SPL. These assets are designed for reuse, such that the development of individual family members can be achieved with reuse from the assets in a prescribed way. Although the above quoted description spotted the need for developing family of programs via code reuse, Parnas continued to discuss the importance of anticipating changes before one begins the design [121], i.e., during requirements engineering (RE) activities. To that end, code reuse is to a SPL what icing is to a cake – a welcome addition, but not the main attraction [35].
In this chapter, we survey a number of existing RE approaches for SPLs. Our survey spans the core RE activities: plan and elicit, model and analyze, communicate and agree, and realize and evolve [115], with special emphasis on how commonalities and variabilities of the product line are handled. We begin with an overview of some important assumptions and considerations that are essential to understand the role of RE in SPL engineering (Section 2.1). Then, in Sections 2.2 through 2.5, we survey several approaches organized by the core RE activities. We conclude this chapter in Section 2.6 by summarizing the surveyed approaches and by pointing out the challenges this thesis addresses in order to lower the SPL engineering adoption barrier.

2.1 The Role of RE in SPL Engineering

2.1.1 SPL Philosophy

There is probably no better way to think of the philosophy of a product line than by relating it to family genetics. Obviously, the genes relate the group of persons, and the individuals also often share the same family name, the same residence, and, sadly, even the same family disease. We tend to see them as a single, unified unit in many circumstances, but remember that they do differ from each other. They normally have different appearances, ages, characteristics, and behaviors. One is more closely related to some than to others. And some may form their own families . . . The analogy continues, yet here we must stop and provide ourselves one of the clearest definitions of a SPL:

“A software product line is a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way.” [35]
This definition is attractive for a number of reasons. First, it highlights the underlying domain specificity of any SPL. Unlike traditional RE activities that explore a quite open problem world, scoping is so crucial in RE for SPLs that we shall always try to restrict our interests within the defined product line boundary. In other words, everything in a SPL is domain-dependent. Scope, commonality, and variability (SCV) analysis [37] is one of the main themes that cut across all SPL engineering activities.

Second, the definition spells out the importance of development in a SPL. In fact, RE for SPLs is about making decisions or making it possible to postpone decisions till a later stage. The decisions made in this phase are not necessarily optimal ones, but the ones that will drive the development of both the entire family and individual products. We thus take economic advantage of the fact that many of the products we produced are very similar – not by accident, but because we planned it that way. We make deliberate, strategic decisions and are systematic in effecting those decision [35].

Finally, the definition refers to the set of core assets. In mature product line organizations, the concept of multiple products disappears. Each product is simply a tailoring of the core assets. It is the core assets that are designed carefully and evolved over time. It is the core assets that are the organization’s premier intellectual property [35]. This mandates the construction and evolution of the core assets as the basis for products’ development. This very benefit of SPL, dialectically speaking, becomes its own burden in that it is not usually worth building the set of core assets by itself. This set is not usually a product that anyone would ask or pay for.

To close our discussion of SPL philosophy, let’s look back at the analogy made to family genetics. Taking any analogy blindly would pose some inappropriateness. An example follows: while the development in a SPL tends to be prescriptive, that in a family involves too many unforeseeables. Nevertheless, we use “software product line” and “software family” interchangeably in this thesis. This is also compliant with the description of “program families” [120] cited in the beginning of this chapter.
2.1.2 Requirements as Core Assets

The goal of SPL engineering is to support the systematic development of a set of similar software-intensive systems by understanding and controlling their common and distinguishing characteristics. In a SPL approach, reuse is planned, enabled, and enforced. And it is through the set of core assets that reuse is achieved. The asset base includes those artifacts in software development that are most costly to develop from scratch – namely, the requirements, domain models, software architecture, performance models, test cases, components, and other elements through domain analysis, domain design, and domain implementation processes.

In a definitive survey of software reuse, Krueger pointed out that the effectiveness of a reuse technique can be evaluated in terms of cognitive distance – an intuitive gauge of the intellectual effort required to use the technique [82]. Krueger made it clear that cognitive distance is not a formal measurement that can be expressed with numbers and units. Rather, it is an informal notion that relies on intuition about the relative effort required to accomplish various software development tasks. Thus, one important truism about software reuse is that: For a software reuse technique to be effective, it must reduce the cognitive distance between the initial concept of a system and its final executable implementation [82].

A system’s initial concepts are usually formulated in the requirements by carrying out certain RE activities. RE is concerned with the real-world goals for, functions of, and constraints on software systems [170]. Compared to reusable source code counterparts, reusable requirements place a greater emphasis on the concepts in the problem domain and place less emphasis on the system implementation. This shift in emphasis helps reduce the cognitive distance between the informal requirements of a system and its executable implementation by isolating the software developer from the source-code-level detail [82]. In such a case, the cognitive distance is reduced in the software development life cycle – up from the level that describes how the system is implemented to the level
that specifies what the stakeholders required in the first place.

Most SPL approaches therefore advocate the importance of embodying requirements in the asset base. It is argued that the nature of a product line is to manage the commonality and variability of products by means of a “requirements engineering – change management” process [35]. Three points are worth noting when developing a SPL’s requirements assets: analyzing SCV, documenting the family specification, and capturing architecturally significant requirements.

1. SCV analysis [37], an underlying theme for SPLs, must be performed in requirements phase. In other words, there must be some mechanism by which the complete set of requirements for a particular product (common plus unique) can be produced quickly and easily, implying that the product-specific requirements are stored as a set of deltas relative to the product-line-wide requirements specification.

2. The product-line requirements specification (PRS) [48] provides a single document characterizing the family as a whole. The PRS also provides a place to record decisions that pertain to the family but not to any particular member of the family – e.g., how the family is likely to evolve over time or the order in which the ability to generate different family members should be implemented [48].

3. In product development context, the role of requirements specification shifts from an implementation-independent contract between a system provider and a customer to a reusable asset that are not at all expected to be free of implementation bias. The development of members in a product line emphasizes on satisfying a few primary requirements. These requirements, that are also called architectural drivers, are the most important requirements, which guide the selection between various design options [85].

One of the key purposes of treating requirements as product line’s core assets is to promote effective reuse. The idea of software reuse is simple: use of existing software or
software knowledge to construct new software [51], yet applying this idea systematically is a challenge. As a status report indicated over a decade ago, domain analysis and domain engineering may help us finally achieve systematic, formal, and effective reuse [130]. Next, we discuss the relationships between RE and domain analysis.

2.1.3 RE and Domain Analysis

Domain analysis was introduced by Jim Neighbors in 1980’s as the activity of “identifying objects and operations of a class of similar systems in a particular problem domain” [103]. It is the process of identifying, collecting, organizing, analyzing, and representing a domain model and software architecture from the study of existing systems, underlying theory, emerging technology, and development histories within the domain of interest [128]. More specifically, domain analysis deals with the development and evolution of an information infrastructure to support reuse. Components of this infrastructure include domain models, development standards, and repositories of reusable components. Domain and boundary definitions are also activities of domain analysis.

Figure 2.1 shows the SADT [137] context diagram proposed in [128], attempting to provide a unified view of domain analysis. The left, right, top, and bottom of the box in Figure 2.1 represent the inputs, outputs, mechanisms, and controls involved in domain analysis respectively. Information is collected from existing systems in the form of source code, documentation, design, user manuals, and test plans, together with domain knowledge and requirements for current and intended systems. Domain experts and domain analysts extract relevant information and knowledge. They analyze and abstract it. With the support of a domain engineer, knowledge and abstractions are organized and encapsulated in the form of domain models, standards, and collections of reusable components. The process is guided by domain analysis methods and techniques as well as management procedures [128].

Domain analysis is key to vertical reuse, whose goal is to derive generic models for
families of systems that can be used as templates for assembling new systems. It is through domain analysis that knowledge is transformed into generic specifications, designs, and architectures. Generic templates will be the basis for creating components that are easy to reuse [130].

Note that domain analysis is only one of the three ingredients of domain engineering. The other two are domain design and domain implementation. Domain engineering is a process for creating a competence in application engineering for a family of similar systems. Domain engineering covers all the activities for building software core assets. These activities include identifying one or more domains, capturing the variation within a domain (domain analysis), constructing an adaptable design (domain design), and defining the mechanisms for translating requirements into systems created from reusable
Figure 2.2: Domain engineering and application engineering processes

components (domain implementation).

The activities of domain engineering play a role in the design of domain-specific languages (DSLs) [98] and SPLs. The whole point of doing domain engineering is to make application engineering easy and cost-effective. Figure 2.2 shows typical processes of domain engineering and application engineering in the context of SPLs [159].

It is clear from Figure 2.2 that RE for SPLs and domain analysis both deal with problem domain concepts. Domain design and implementation, on the other hand, focus more on the actual development of reusable assets. An overview of domain analysis was given in [128], where the intertwining of problem and solution domain concepts is more evident than in RE for a single system [114].

The differences between RE and domain analysis are also noticeable. While domain analysis aims at discovering and documenting all the knowledge of the domain, the primary role of domain knowledge in (single-system) RE is in supporting refinement of requirements to implementable specifications [172].
In RE for singular one-of-a-kind software systems, it is the specification that bridges RE and software development activities. The output of formal domain analysis is a domain model consisting of: 1) a domain definition defining the scope of the domain, 2) domain terminology (vocabulary, ontology), 3) descriptions of domain concepts, and 4) feature models describing the commonalities and variabilities of domain concepts and their interdependencies [98]. Therefore, in RE for SPLs, it is the specification for the product family that bridges RE and product line development activities.

Despite RE and domain analysis have different perspectives and focuses on the processes and artifacts, many people view domain engineering as equivalent to product line engineering, thus view domain analysis and RE for SPLs as intimately intertwined concepts [51].

2.2 Plan and Elicit

2.2.1 Two-Tiered Organization

SPL practices affect many parts of an organization. Roles such as product planning will change from a single-product focus to working with a set of related products and can take advantage of the SPL approach. Planning a set of products at one time as opposed to individually over time produces economies of scale.

The degree to which organizational changes must occur before delivering the first product varies. In heavyweight approaches, the organization assigns specific teams to produce assets such as the architecture and components. In lightweight approaches, the organization might create the first few products as usual and then use mining efforts to extract SPL assets after the fact.

Numerous organizations in various industries have reaped significant benefits using a SPL approach for their systems. Despite this diversity, the SEI has distilled universal and essential SPL activities and practices [35]. At the highest level of generality are
three essential and highly iterative activities that blend technology and business practices. Fielding a SPL involves \textit{core asset development} and \textit{product development} using the core assets under the aegis of technical and organizational \textit{management}. Core asset development is often referred to as \textit{domain engineering}, and product development as \textit{application engineering}. Figure 2.3 illustrates this triad of essential activities.

Figure 2.3 shows that an organization using product line practices is structured to facilitate two fundamental roles: development of reusable assets and development of products that use those assets. The first role includes all products while the second focuses on a single product. Organizations accomplish this division of responsibility in a variety of ways — some have teams dedicated to each role and others use the same people for both.

Recent research on methods behind a new generation of SPL successes pointed out that the “domain engineering – application engineering” dichotomy has many negative effects, including a cultural “us-versus-them” tension and diverging product software branches [84]. The proposed solution was to shift the balance away manual application engineering and focus entirely on domain engineering and fully automated production
using SPL configurators [84].

No matter how much application engineering should be de-emphasized, core asset development and product development in a SPL are inextricably intertwined. The rotating arrows in Figure 2.3 indicate not only that companies use core assets to develop products but also that revisions of existing core assets or even new core assets might (and most often do) evolve out of product development [113]. In some contexts, organizations mine existing products for generic assets – perhaps a requirements specification or an architecture – that they then migrate into the SPL’s asset base. In other cases, the core assets might be developed or produced for later use in product production. There is a strong feedback loop between the core assets and the products. Core assets are refreshed as organizations develop new products. They then track asset use, and the results are fed back to the asset development activity. Technical and organizational managers manage this process carefully at all levels [113].

### 2.2.2 Scoping

The key to the enterprise-level strategic positioning is understanding the scope of the SPL. A product line’s scope states what systems an organization would be willing to build as part of its SPL and what systems it would not. In other words, it defines what’s in and what’s out.

Clements refers scoping as drawing a doughnut in the space of all possible systems [34]. The doughnut’s center represents the set of systems that the organization could build, and would be willing to build, under the auspices of its product line capability. Systems outside the doughnut represent those that are out of scope, that the SPL is not equipped to handle well. Systems on the doughnut itself could be handled with some effort, but require case-by-case disposition as they arise [34]. For example, in a SPL of office automation systems, a product with a conference room scheduler would be in, but one with a flight simulator would be out. One with a specialized intranet search engine might be
Table 2.1: Software product line scoping

<table>
<thead>
<tr>
<th>Roles</th>
<th>Levels</th>
<th>Activities</th>
</tr>
</thead>
<tbody>
<tr>
<td>scoping expert</td>
<td>product portfolio scoping</td>
<td>product line mapping</td>
</tr>
<tr>
<td>domain experts</td>
<td>domain scoping</td>
<td>domain potential assessment</td>
</tr>
<tr>
<td>SPL manager</td>
<td>asset scoping</td>
<td>reuse infrastructure scoping</td>
</tr>
</tbody>
</table>

in if it could be produced in a reasonable time and if there were strategic reasons for doing so (such as the likelihood that future customers would want a similar product).

The purposes of product line scoping, among others, include: (a) examining existing product, (b) understanding SPL goals, (c) developing SPL scenarios, and (d) identifying innovative functionality for the future [35]. Recent research defines a few dimensions that one must consider in scoping a SPL [72], as summarized in Table 2.1.

The roles identified for the overall scoping process are a scoping expert, domain experts with technical or marketing knowledge, and the SPL manager. The scoping experts are one or more persons that drive and customize the scoping activities and conduct workshops and interviews. They should not be part of the development team as they should push an external view on the products and domains. The domain experts provide their knowledge on the products and the application domain from technical or marketing side and the SPL manager provides an overview on the SPL, its goals and embedding in the organizational environment [72].

Three levels of scoping can be identified [72]:

- **Product portfolio scoping** aims at determining which products should be included in the SPL and what features they should provide.

- **Domain scoping** aims at identifying and bounding the domains, i.e., areas of functionality, that are important to the SPL and provide sufficient reuse potential.
Asset scoping aims at determining specific assets to be developed for reuse. The combination of all these assets then forms the reuse infrastructure.

Three activities are highlighted in the scoping process [72]:

- Product line mapping identifies the products and features of a SPL and builds a *product feature matrix* as central artifact. The goal of this phase is to obtain an overview on the SPL and its features and their distribution in the products [142].

- Domain potential assessment evaluates the sub-domains of the application domain of the SPL for their reusability and product potential. The goal of this phase is to identify domains with a high reuse potential where further activities (like architecting) should focus on.

- Reuse infrastructure scoping identifies the existing and planned assets (requirements, architecture, components, etc.) and quantifies their reuse potential (and so the reuse potential of the whole SPL) with an economic model. The goal of this phase is to plan the SPL infrastructure and to identify development needs.

Explicitly scoping the SPL lets us examine regions in the neighborhood that are underrepresented by actual products in the marketplace, make small extensions to the SPL, and move quickly to fill the gap. The scope feeds other SPL artifacts; the requirements, architecture, and components all take their cues for the variabilities they need to provide from the scope statement [34].

The result of scoping is a scope definition document, which itself becomes a SPL core asset. The scope definition identifies those entities with which products in the SPL will interact (i.e., the PL context), and it also establishes the commonality and bounds the variability of the SPL [35].

Note that the SPL scope may not come into sharp focus all at once. The market analysis for the SPL will usually contain a fuzzy description of the scope. A refined
version emerges as domain understanding increases. The detailed modeling of user-visible services during a SPL requirements stage completes the SPL scope. Scoping does not end here, but is refined as the SPL architecture and components are developed. Therefore, beyond the initial key factors listed in Table 2.1, scoping is a continuous process that monitors the developments in all relevant markets, application or technical domains, and customer request.

2.2.3 Elicitation

Table 2.2 classifies various requirements elicitation techniques into traditional techniques, collaborative techniques, cognitive techniques, and contextual approaches. Generally speaking, these techniques can all be applied to elicit SPL requirements. In this section, we focus our discussion on specific RE frameworks tailored for SPLs. We then summarize
the elicitation techniques commonly used in SPLs.

1. Feature engineering

The field of feature engineering has worked on modeling domain constructs and their relationships for a number of years. For FODA [75], a feature is “a prominent or distinctive user-visible aspect, quality or characteristic of a software system or systems”, whereas for feature programming [12], a feature is “an increment in program functionality”. These imply that when there is no system yet, there cannot be a discussion about features. To make sure we can talk about features before any system is built, Czarnecki and Eisenecker defined a feature as “an important property of a concept instance [which] allow[s] us to express the commonalities and differences between concept instances” [39].

Having pursued feature engineering for a couple of years, Kang [76] pointed out that the use of “features” is motivated by the fact that customers and engineers often speak of product characteristics in terms of “features the product has and/or delivers”. They communicate requirements or functions in terms of features and, to them, features are distinctively identifiable functional abstractions that must be implemented, tested, delivered, and maintained [76].

Such a vision has two major shortcomings. First, it equates features to functional requirements, and ignores non-functional ones or system qualities, such as usability and security, which play an essential role in RE and architectural design. Second, it envisaged that standard terms (features) used by the stakeholders would naturally emerge at the right level of abstraction. Many studies presented contradictory evidence. For instance, engineers working on the LG’s elevator control SPL did not agree on what specific features meant, even after 3 months of domain analysis [35].

In feature engineering, the dominant elicitation and modeling technique is domain analysis [128]. The stakeholders involved in domain analysis include problem do-
main expert, domain analyst, and domain engineer. The sources of domain knowledge mainly come from technical literature, existing implementations, customer surveys, expert advice, and so on. The output is a domain model, a feature model in particular, that contains taxonomies, standards, functionalities, etc [128].

One drawback of domain analysis refers to its intrinsic domain dependence. The process counts on experts’ experience and intuition to manually acquire domain knowledge. Namely, there are no rules that enable engineers to identify domain elements easily [100].

2. Use case modeling

A use case describes the communication between an actor and a system for the exchange of information, as well as the actions that must be carried out internally by the system to respond to these information requests. Although use cases are fundamentally a text form [36], researchers tend to exploit the graphical representations in SPL engineering.

Halmans and Pohl realized that the variability of the SPL must be adequately considered when eliciting requirements from the customer [59]. They differentiated between variability aspects which are essential for the customer and aspects which are more related to the technical realization and need thus not be communicated to the customer. Since use cases represent user-visible behavioral requirements of the system, they proposed extensions to use case diagrams, mainly variation points and variants, to support an intuitive representation of customer relevant variability aspects.

Although the elicitation of (extended) use cases was not explicitly addressed in [59], heuristics of identifying domain concepts exist in the object-oriented modeling literature [152]. One can apply the relatively simple noun-extraction technique from the problem statement to determine actors and objects, and apply verb-extraction
3. Primitive requirements (PR)

Domain use cases were developed to model requirements in a SPL [100]. The elicitation of domain use cases was done by means of primitive requirements (PR). A PR is a transaction that has an effect on an external actor.

It was argued in linguistics that any semantically complex word can be explicated by means of an exact paraphrase composed of simpler, more intelligible words than the original [162]. These simpler and more intelligible words are known as semantic primitives. The meaning of requirements can be decomposed into sets of semantics primitives. The divided semantic primitive is defined as the PR. Thus, the term “PR” is used as a building block of a more complex refinement [100].

PRs for a specific domain can be identified by analyzing functional requirements of legacy systems in the domain [100]. Besides, input from different stakeholders and documents of legacy systems can be used to discover the commonality and variability of the PR once the PR is identified. Table 2.3 shows a sample PR-context matrix, in which “O” indicates that the PR is found in the system and “X”

Table 2.3: A PR-context matrix

<table>
<thead>
<tr>
<th>PR</th>
<th>CBC</th>
<th>NY Times</th>
<th>BBC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modify member information</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>Search a scrapbook</td>
<td>X</td>
<td>O</td>
<td>X</td>
</tr>
<tr>
<td>Write an opinion</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
indicates that the system does not have the PR. From Table 2.3, “modify member information” and “write an opinion” are common PRs in the news information repository domain, whereas “search a scrapbook” is an optional PR.

4. Goal modeling

Goals express, at various levels of abstraction, stakeholders’ many objectives for the system under consideration [160]. Goal modeling shifts the emphasis in requirements analysis to the actors in an organization, their goals, and the interdependencies between those goals, rather than focusing on processes and objects. Goal modeling allows us to analyze variabilities from the problem domain perspective, even before any product is actually built.

Systematic effort on acquiring and analyzing variabilities in goal models has been made in [65, 89, 90]. In [65], user goals, skills, and preferences were sequentially analyzed in order to generate a customizable software design. In [89], the “options” of an existing product’s configuration were considered as OR-decomposed subgoals or tasks so that high-level user goals and softgoals can be identified. In [90], a text corpus relevant to the domain of discourse was used to derive problem-specific variability concerns. In particular, linguistic cases, such as agentive, objective, locational, temporal, were applied to define the set of frame elements (semantic slots) associated with each verb in a requirements statement.

This series of goal-oriented work points out two directions of applying variabilities: adaptation and reuse. Adaptation refers to the ability of a software system to support all configurations that may be needed to address a variety of personal and contextual cases, from the user’s perspective. Reuse, as pursued in SPLs, explores ways by which one can define core assets capable of serving as the basis for cost-effective derivation of products for a particular market segment or mission, from the product’s perspective. While SPL engineers tend to consider variabilities in
an application domain, goal modeling offers a complementary, problem-oriented framework for SPL requirements elicitation.

A few more RE frameworks for SPLs will be discussed in Section 2.3. To recap, most elicitation techniques focus on functional or behavioral requirements. Relating to Table 2.2, traditional and collaborative techniques, especially reading existing documents and analyzing hard data, are widely adopted, whereas cognitive and contextual approaches are rarely applied in SPL engineering.

2.3 Model and Analyze

Modeling is an important activity in RE. Desiderata for requirements modeling notations include: implementation independence, abstraction, formality, constructability, ease of analysis, executability, traceability, and minimality. A model is created for a specific purpose and contains information only needed to fulfill its purpose. Recent work suggests that requirements models, which should be palatable to users, could be applied to facilitate communication, uncover missing information, organize information, and uncover inconsistency [64].

Requirements modeling frameworks have different focuses such as modeling organization, modeling information and behavior, or modeling system qualities. In this section, we review six modeling techniques for SPLs.

2.3.1 Goal Modeling

Goal modeling has been found to be an effective way for identifying requirements of software systems by focusing on understanding the intentions of the involved stakeholders [160]. Central to goal modeling is the idea of constructing hierarchies of AND- and OR-decompositions of high-level stakeholder goals and then, recursively, into low-level
subgoals and tasks that lead to requirements of the system-to-be. When a goal is AND-decomposed into subgoals, all of them must be satisfied for the parent goal to be satisfied. When a goal is OR-decomposed, the satisfaction of one of the OR-subgoals suffices for the satisfaction of the parent goal.

The OR-decompositions in a goal model represent stakeholder intentional variability from the problem-domain perspective rather than from the product’s perspective. Figure 2.4 shows a sliced goal model for a meeting scheduler [168, 88]. One way of
systematically identifying variability concerns is analyzing existing textual information specific to the domain and the goal [90]. Thus, each goal can be associated with a set of variability concerns, each of which must be addressed through an OR-decomposition in the AND/OR tree that emerges from the analysis of the goal. In Figure 2.4, each OR-decomposition is annotated with the variability concern that it addresses. For example, alternative agents that can collect the constraints address the *agentive* variability concern, while different algorithms for picking a suitable slot address the concern *manner*. By referring to the variability concerns that are relevant to the goal at the time we decompose it, we allow the discovery of variability that might otherwise remain hidden [88]. Thus, the *dative* concern, which refers to the agent that is affected by the achievement of a goal, is used to suggest that different response styles to an invitation (to everybody or only to the solicitor) may need to be accounted.

Goal models not only support early-phase RE modeling [168], but also facilitate the mappings from requirements to designs and actual products. Note that the annotated goals are variation points, and the OR-decomposed tasks are variants of a particular variability concern. Given a set of variability concerns, some techniques define the fitness criteria that can be used for evaluating alternatives implied by goal models. We discuss three categories of criteria that can be used for selecting alternatives in goal models: user skills, contextual characteristics, and stakeholder preferences.

User skills are associated with leaf level goals in goal models, which describe tasks to be performed by humans alone or through interaction with the system. Thus, for each task, we construct a list of skills that a user who needs to perform the task must have [65]. Conversely, if we are given the skills of a particular individual we are able to filter out courses of tasks that she cannot perform. In Figure 2.4, assume that tasks at the leaf level mention that an invitee responds to the constraint collection by filling up a Web form. This requires some sensory (e.g., vision) and motor or speech skills to perform the input, some basic cognitive skills as well as language and computer skills. Individuals that
lack any of the required skills (e.g., elderly, people with cognitive or other impairments, illiterate) must be provided a different alternative whose tasks they can actually perform. This may even imply that a completely different strategy for collecting constraints must be selected at the higher level [65].

A similar approach is followed for contextual characteristics: tasks can only be performed under specific circumstances [90]. In the example of Figure 2.4, constraints can be selected from individual calendars only if the invitee has made them public and there is a means to access them through a network. From a modeling point of view, contextual characteristics are a generalization of skills, as they are also used as conditions for the performance of leaf level tasks [88].

User preferences, on the other hand, are specifications of priorities over softgoals that alternatives must satisfy [65]. As opposed to hard goals, softgoals are goals for which there is no clear criterion that can be used for deciding whether they are satisfied or not. In Figure 2.4, softgoals are represented as cloud-shaped elements. Contribution links, annotated with “+” and “-” symbols in the figure, allow us to represent the fact that satisfaction of hard goals affects positively or negatively the satisfying of softgoals. Thus, in Figure 2.4, the choice to broadcast one’s constraints may hurt one’s privacy, but it helps building a participatory spirit. Conversely, if we state that building a participatory spirit is more important than privacy of individual invitees, we implicitly suggest that the particular response style should be used. Thus, by specifying the relative importance of softgoals, we implicitly bind low level options.

2.3.2 Use Case Modeling

Use cases are typically being documented using two representations: use case diagrams and use case templates. Whereas the UML 2.0 defines the syntax and a vague semantic for use case diagrams, it does not propose a template for documenting use cases and their scenarios. However, several authors have proposed use case templates, e.g., [36].
There are two camps to apply use case modeling in a SPL context: building domain use cases as core assets [100] and communicating variability to customers [59]. In both cases, use cases for a SPL are documented graphically with some extensions to standard notations. The main reason is that it is relatively easier to convert a use case diagram to the template representation than to mine text to generate a graphical model.

In [100], a straightforward way of introducing SPL use cases was applied: stereotypes. Using this mechanism, one could annotate common (mandatory) use cases and variable (optional) ones. Figure 2.5 shows a domain use case diagram for news information repository domain [100]. The use cases were identified via primitive requirements (cf., Section 2.2), and the common/variable property of each use case was determined by a PR-context matrix (cf. Table 2.3). Since the only extension that a domain use case model makes is the common/optional stereotype, the relationships between use cases (include, extend), between actors and use cases (association) remain the same as in UML standard.
A serious problem of using stereotype is that variation points are still invisible [59]. For example, in Figure 2.5, the two options “Give Opinion” and “Scrap Article” are non-mutually-exclusive actions an actor can perform on the “Search News” results. The variants are visible, but the variation point – feedback actions on search results – are not.

To explicitly model variation points, extensions to use case diagrams are proposed in [59] to represent essential functional variability, as opposed to technical one, that can be communicated to customers directly. Figure 2.6 shows the extensions to model variation points and variants. A triangle represents a variation point and is associated (included) with a use case. Dark and grey triangles represent mandatory and optional variation points respectively. Mandatory means a decision has to be made at this point, whereas optional does not require a definite answer at this point. Variant use cases are attached to certain variation point, with cardinality and mandatory/optional (darked/white circle)
property specified.

It should be noted that the main purpose of product-line use case modeling is to document variability and communicate it to customers. Since standard notations do not allow us to model variation points or variants, extensions to use case diagrams were introduced.

2.3.3 Feature Modeling

Feature models are a popular means for documenting SPL requirements. The seminal work of FODA was introduced in [75]. Since this initial proposal, several extensions of feature models have been devised in the SPL literature, all of which have the built-in notions of mandatory, optional features and some dependencies between the features, such as requires and excludes, in order to model the commonalities and variabilities in a SPL. Some feature models are trees, and some are directed acyclic graphs (DAGs). In all dialects, nodes represent features, and the root node represents the concept. If a feature has children (sub-features), they are connected to the parent via decomposition links (solid edges). Feature dependencies are annotated by dashed arrows in the diagram. We refer to [144] for a recent survey on feature modeling.

One purpose of feature modeling is to show how features can be combined together to derive a concrete product. Unfortunately, most feature models lack a formal semantics, which makes interpreting a feature diagram ambiguous. Some people appreciate this lack of semantics. For example, it was argued that the edges in a feature diagram must not have any complete semantics of their own so that the configurability aspect is separated from other aspects [39]. Thus, for example, feature models cannot have structural semantics, like consists-of or is-part-of. A feature model shall be seen only as a visual way to represent a set of constraints that need to be satisfied when creating subsets of features.

Although the edges by themselves may not have any semantics, the semantics of a fea-
The feature model must be defined to enable unambiguous interpretation and proper reasoning. Current approaches seldom deal with the semantic issue. As an example, consider the two feature models in Figure 2.7. The left one follows FODA notations [75], where every feature is mandatory and \((l, m)\) and \((m, n)\) are alternative/XOR sub-features of \(x\) and \(y\) respectively. The alternation relates to nodes in the diagram, so this model has products (valid configurations) \(\{r, x, y, l, n\}\) and \(\{r, x, y, m\}\). The right model in Figure 2.7 follows the notations defined in [134], where alternation is edge-based. Therefore, in addition to the two products defined by the left model, \(\{r, x, y, l, m\}\) and \(\{r, x, y, m, n\}\) are valid configurations defined by the right feature diagram in Figure 2.7.

To avoid ambiguities and to start building safe automated reasoning tools, formal semantics of feature diagrams must be defined. One such endeavor came from [144]. The syntax of a feature diagram is given as follows.

**Definition 2.3.3.1** The preliminary constructs for defining a general syntax of feature diagrams are:

- **\(NT = \{\text{and, or, xor, card}\}\).** Node type \(NT\) defines the set of operators that can be performed on a node; and

- **\(GCT = \{\text{requires, excludes}\}\).** Graphical constraint type \(GCT\) is either \textit{requires}
or excludes.

**Definition 2.3.3.2** A feature diagram is a tuple \( f_d = (N, r, t, DE, CE) \) that satisfies the listed axioms, where

- \( N \) is the set of nodes of \( f_d \);
- \( r \in N \) is the root/concept of \( f_d \);
- \( t : N \rightarrow NT \) labels each node with an operator from \( NT \);
- \( DE \subseteq N \times N \) is the set of decomposition edges; \( (n, n') \in DE \) will rather be noted \( n \rightarrow n' \); and
- \( CE \subseteq N \times GCT \times N \) is the set of constrained edges.

The feature diagram \( f_d \) must satisfy the following axioms:

1. Only \( r \) has no parent: \( \forall n \in N . (\neg \exists n' \in N \text{ s.t. } n' \rightarrow n) \iff n = r. \)

2. \( DE \) is acyclic: \( \neg \exists n_1, \ldots , n_k \in N \text{ s.t. } n_1 \rightarrow \ldots \rightarrow n_k \rightarrow n_1. \)

It is shown that the above abstract syntax covers seven commonly used feature diagrams in the literature [144]. Then, a formal semantics is given by a function from the syntactic domain of a language (Definitions 2.3.3.1 and 2.3.3.2) to a semantic domain, which is given as follows:

**Definition 2.3.3.3** A model of a feature diagram \( f_d = (N, r, t, DE, CE) \) is a subset of its nodes: \( m \in \mathbb{P}N \). A model \( m \) is valid for \( f_d \), noted \( m \models f_d \), if and only if:

1. The concept is in the model: \( r \in m. \)

2. The model must satisfy all graphical constraints: \( \forall (n_1, gct, n_2) \in CE. gct(n_1 \in m, n_2 \in m) \) must be true, where \( gct \in GCT. \)
3. If $s$ is in the model and $s$ is not the root, one of its parents $n$ must be in the model:

$$\forall s \in N.s \in m \land s \neq r : \exists n \in N : n \in m \land n \rightarrow s.$$ 

The benefits of defining a formal semantics of feature diagrams include gaining a shared understanding, comparing feature languages, implementing decision procedures, and so forth. We refer to [144] for more information about this rigorous treatment of feature modeling. Note that the formalism is not a user notation meant to be used by analysts to draw feature diagrams. It is a framework to be used by method engineers to formally define, study, and compare feature languages. It also provides a formal foundation for building automated tool support for feature modeling.

### 2.3.4 Propositional Formulas

Feature models are used to specify members of a SPL. Despite years of progress, tools for feature models often seem *ad hoc*; they provide limited support for feature constraints and offer little or no support for debugging feature models. Debugging means detecting and repairing inconsistencies in the feature model. To address the weaknesses in existing feature model tools and theories, Batory integrated prior results to connect feature models, grammars, and propositional formulas [12]. This work not only provides the fundamental underpinnings of feature models, but also enables off-the-shelf satisfiability solvers to debug feature models. In this section, we briefly review the connection between feature models and propositional formulas.

The connection between feature diagrams and grammars serves two purposes: a) to provide a graphics-neutral representation of feature models, and b) to map feature models to propositional formulas. In [12], feature diagrams are described via an iterative tree grammar, which uses iteration rather than recursion to express repetition, and requires every token/terminal to appear in exactly one pattern and the name of every production to appear in exactly one pattern. The root production is an exception; it is not referenced in any pattern.
Figure 2.8: A feature diagram and its grammar

Figure 2.9: GUI specification

Figure 2.8 shows a feature diagram and its grammar. The feature diagram defines a product line where each application contains two features \( r \) and \( s \), where \( s \) is an alternative feature: only one of \( G \), \( H \), and \( I \) can be present in an application. \( s \) is a compound feature that consists of mandatory features \( A \) and \( C \), and optional feature \( B \). Figure 2.8b is the grammar of Figure 2.8a. An application/product defined by the feature diagram of Figure 2.8a is a sentence of the grammar listed in Figure 2.8b.

Grammars provide a graphics-neutral representation of feature models. For example, the grammar of Figure 2.8b could be displayed by the feature diagram of Figure 2.8a or the GUI specification of Figure 2.9. Note that the GUI does not display features \( A \) and \( C \), as they are mandatory – nothing needs to be selected. GUI specifications like
Figure 2.9 can be used for staged configuration, which presents an incremental way to progressively specialize feature models [39].

We now show how to convert the grammar of a feature model to propositional formulas. A propositional formula is a set of boolean variables and a propositional logic predicate that constrains the values of those variables. Besides the standard $\land$, $\lor$, $\neg$, $\Rightarrow$, and $\Leftrightarrow$ operations of propositional logic, we also use $\text{choose}_1(ex_1, \ldots, ex_k)$ to mean at most one of the expressions $ex_1 \ldots ex_k$ is true. More generally, $\text{choose}_{n,m}(ex_1, \ldots, ex_k)$ means at least $n$ and at most $m$ of the expressions $ex_1 \ldots ex_k$ are true, where $0 \leq n \leq m \leq k$.

A grammar is a compact representation of a propositional formula [12]. We use the grammar of Figure 2.8b as an example to illustrate how to generate formulas from productions.

- **root.** The root is always true. For example, $e = true$.

- **and-decomposition.** For example, the production “$e : r s$” is expressed as “$e \Leftrightarrow r \land e \Leftrightarrow s$”.

- **alternation.** For example, the production “$r : G \mid H \mid I$” is expressed as “$r \Leftrightarrow \text{choose}_1(G, H, I)$”.

- **option.** For example, the production “$s : A [B] C$” is expressed as “$s \Leftrightarrow A \land B \Rightarrow s \land s \Leftrightarrow C$”.

In addition, the conversion from graphical constraints \{requires, excludes\} to propositional formulas is straightforward. The main benefit of formulating a SPL’s feature model in propositional formulas is to leverage efficient off-the-shelf tools (e.g., SAT solvers) to bring useful capabilities to existing reasoning and analysis tools. Some progresses along this line of research are reported in [12, 99].
2.3.5 Product-Line Requirements Specification (PRS)

In a seminal paper on the meaning of requirements, Jackson described the relationship between user requirements ($R$), domain assertions ($E$), and specification ($S$): $E, S \vdash R$ [67]. This tells us to find the right $S$ in the equation, given $E$ and $R$. In other words, $S$ is the bridge between RE and SE [67], and is often regarded as one of the most important products in software development.

Faulk proposed to develop the product-line requirements specification (PRS) for a SPL, much like to develop the software requirements specification (SRS) for a single software system [48]. Having a distinct PRS offers potential advantages:

- The PRS provides a single document characterizing the family as a whole for domain engineers and other stakeholders interested in the product line.

- The PRS provides a place to record decisions that pertain to the family but not to any particular member of the family.

- Individual SRS for a family member can be derived from the PRS, which provides a basis for rapid product development.

To address the needs particular to a PRS as well as those for any derived SRS, PRS objectives include specifying commonalities and variabilities in the SPL and supporting requirements reuse and analysis.

In [48], the PRS of a flight control SPL was illustrated using SCR notations and textual representations. The PRS was developed following domain analysis [37]. Output of the analysis is a written specification of the SPL’s terminology, commonalities, variabilities, and dependencies.
Chapter 2. Background

2.4 Communicate and Agree

Communicating and agreeing requirements among stakeholders can improve the understanding of the problem domain, uncover inconsistencies, and push software development forward. Two aspects of an agreement need to be considered: validation and negotiation [41]. Validation can be achieved by prototyping or by reviews and inspections. Negotiation involves prioritization and conflict resolution. In this section, we review three SPL requirements methods from the communication perspective: feature modeling, use case modeling, and definition hierarchy.

2.4.1 Terminological Interference of Features

The original proposal of feature orientation envisaged that standard terms (features) used by the stakeholders would naturally emerge at the right level of abstraction [75], but many studies presented contradictory evidence. For example, engineers working on the LG’s elevator control SPL did not agree on what specific features meant, even after three months of domain analysis [35]. Experience also showed that if two features simply had different marketing names, they were often mismatched and developed separately [40].

Even though some research worked on the semantics of feature models, little attention has been paid to the semantics of features themselves. In [144], a feature is just the label of a box in the diagram. However, features to many stakeholders are abstract concepts, embodying a set of related primitive elements. Mismatches in stakeholders’ abstraction vocabulary often occur, as an area of surprising controversy. Experts would occasionally misunderstand one another, because they were using the same words in different ways. In fact, experts would sometimes be in violent agreement with one another, all the while expressing the same idea in different terms [35].

In our earlier work [105, 107], we discovered that stakeholders do not use their terminologies consistently when stating requirements. We applied George Kelly’s repertory
grid technique (RGT) [52] to detect stakeholders’ inconsistent use of terminology when expressing abstract concepts. One of the RGT’s strengths is that it avoids the problem of imposing an unnatural terminology on stakeholders – it essentially treats a term’s meaning as a relationship between signs and actions.

Our experience showed that establishing a shared common ground (context) among stakeholders is a preliminary step to study the terminological interference problem, but that left as an open question in the earlier work. One of our proposed research is to search for primitive elements to form the ground for aligning abstract concepts (e.g., features and non-functional requirements) used in SPL engineering. In particular, we are interested in the notion of “primitive requirements” (PRs). It was argued in linguistics that any semantically complex word can be explicated by means of an exact paraphrase composed of simpler, more intelligible words than the original. In [100], the authors introduced the idea of using PRs in SPLs. However, the identification of PRs was implicit in [100]. We plan to leverage techniques in information retrieval [140], natural language processing [73], and goal-based variability analysis [90] to identify and represent PRs.

2.4.2 Communicating Variability to Customers

Variability is a central concept in SPL development. Variability empowers constructive reuse and facilitates the derivation of different, customer specific products from the SPL. If many customer specific requirements can be realized by exploiting the product line variability, the reuse achieved is obviously high. If not, the reuse is low. It is thus important that the variability of the SPL is adequately communicated to the customers.

In [58], the need to communicate the variability of the SPL to the customer was emphasized. Two types of variability of a SPL are classified: essential variability and technical variability. Essential variability describes the variability of the SPL from a customer/user viewpoint, i.e., from a system usage perspective. An essential variability does not contain technical realization aspects. Essential variability includes functionality,
system environment, quality, data, etc. Technical variability deals with aspects about the realization/implementation of the variability and/or if it states consequences on the IT-infrastructure. Thus, technical variability includes IT-infrastructure, binding time, implementation, etc.

To effectively communicate essential variability of the SPL to the customer, the authors proposed to exploit use cases as means for communicating [58], motivated by the successful usage of use cases and scenarios in single product development [36]. Two concepts need to be made explicit in use cases: variation point and variant. Since current use case notations do not support the representation of these essential variability concepts, extensions to use case diagrams are proposed in [58]. The detailed modeling mechanisms are discussed in Section 2.3 (cf. e.g., Figure 2.6). The purpose of these extensions is to support an intuitive representation of customer relevant variability aspects.

2.4.3 Definition Hierarchy for Prioritization

Quality of a software system can be defined as the ability to fulfill stakeholder needs. Because most of the quality attributes emerge from the structuring principles of the whole system, it will be very difficult to change these properties after the system is completed. Such a change can affect the software as a whole – components and their communication mechanisms, their scheduling, roles, interfaces, and data allocation techniques.

It is therefore important to find out the quality attributes and carefully prioritize them. Often real life system development emphasizes on satisfying a few primary requirements. These requirements, called architectural drivers, are the most important requirements that guide the selection between various design options. One has to carefully define these drivers because any change in architectural drivers is very likely to affect the entire architecture. In the SPL development, the architectural drivers are common for the entire product family. They define the reference architecture. Any change in the reference architecture will propagate to all members. The role that architectural drivers
and quality attributes have in the development of a SPL suggests that they should be used as the top most elements in requirements structuring.

In [85], a requirements structuring method – a definition hierarchy – was proposed to help identify architectural drivers of the SPL and show how different products in the family vary. Most methods divide requirements into functional requirements and non-functional requirements. A definition hierarchy divides requirements into design objectives and design decisions. Design objectives are the essence of a requirements document. They state goals and objectives that shall be satisfied during system design. Design decisions are a reflection of the solution domain to the requirements analysis phase. Although in theory we should structure pure requirements without thinking any problems associated with implementation, in practice, we often need a mechanism to get feedback from implementation alternatives to requirements specification in order to keep requirements feasible [114]. In the SPL development process, this feedback is very strong. If we want to take care that the new product is a member of an existing SPL, we must assure that it conforms to the family architecture. In other words, it obeys most of the design decisions underlying that architecture [85].

When structuring requirements, a definition hierarchy creates separate sections for quality attributes such as reliability and security. In the hierarchy, design objectives are defined by other design objectives or design decisions. The hierarchy is a logical AND tree, i.e., the child requirements are used to define the meaning of the parent requirements. Topmost nodes in such a hierarchy represent the architectural drivers and other quality attributes that the system is supposed to fulfill. A definition hierarchy consists of nodes that represent design objectives and design decisions. These nodes are connected with edges that represent refinement when they connect two design objectives. When an edge is between a design objective and a design decision it shows that this requirements is (partially) satisfied by design decisions.

Each node in the definition hierarchy gets a priority that reflects the importance
of that node in supporting the intention of its parent. A node cannot have a higher priority than its parent. In Figure 2.10, three products (A, B, C) of the weather station product line are represented in a definition hierarchy, which focuses on reliability – one of the architectural drivers. Requirements prioritization allows variations among the definitions of super concepts [85]. The lowest value in the scale is irrelevant. This value is used to represent design objectives or decisions that do not belong to the description of that product. A node with irrelevant priority does not define its parent, and irrelevance is expressed by priority zero in Figure 2.10, e.g., system duplication (REQ.1.1) is irrelevant to product A, but is very important to products B and C.

Structuring requirements into a definition hierarchy has many benefits: it provides
concise specification of a general design objective, it supports decomposition and refinement of quality attributes, it relates requirements to design decisions that drive the architecture. There are also some shortcomings of such a requirements structuring method. First, structures based on one dominant decomposition, in this case the quality attributes, cannot modularize tangled and scattered crosscutting concerns. Second, quality attributes are often intertwined and conflicting with each other. Detecting and resolving trade-offs are hard to achieve in the definition hierarchy. Finally, requirements dependencies are not explicitly expressed in the current hierarchical definition.

To sum up the existing literature, communicating and agreeing SPL requirements need to focus on aspects such as commonalities, variabilities, decision points, primitive shared common ground, observable phenomena, architectural drivers, hierarchical structure, and requirements priorities.

2.5 Realize and Evolve

In this section, we review two requirements-based SPL realization methods: feature engineering and product-line requirements specification (PRS).

- **FORM.** FORM (Feature-Oriented Reuse Method) [76] extends FODA [75] to the software design and implementation phases and prescribes how the feature model is used to develop domain architectures and components for reuse. The underlying philosophy for this extension is that the features of a domain characterize each variant product in the domain, and the code that implements the characterizing features should be packaged, managed, and reused as software modules.

  FORM first defines 4 levels for feature refinement: capabilities, operating environments, domain techniques, and implementation techniques. FORM then maps features to a reference architecture, which has three views [33]: a subsystem model, a process model, and a module model. Guidelines for developing each model are
given in [76]. A subsystem model defines the overall system structure by grouping functions (the most important focus of attention) into subsystems that can be allocated to different hardware. A process model represents the dynamic behavior of each subsystem, e.g., concurrency within a subsystem. A module model’s hierarchy reflects the feature hierarchy. In FORM, the architecture models, features, and module implementation correspond to the context, concept, and content [33], respectively.

The application engineering in FORM is feature-driven. The user needs to select features in the feature model (iteratively) so that design and implementation decisions can be derived from the SPL reference architecture, to which the constraints on features are conformed. Other feature-oriented implementation techniques exist, e.g. [11, 39]. Since they do not necessarily follow a well-defined RE process, we will not review them in this section.

- **PRS.** In Section 2.3, we reviewed how to model a SPL’s requirements via product-line requirements specification (PRS). We now study three basic meta-text constructs proposed in [48] in order to derive variations from the PRS.

1. Text replacement variables: a distinguished symbol that is used in the family specification to denote a particular variant. On instantiation of a family member, it is replaced by one of its possible values. For example, in an auto-marker SPL, “course name” is a variable that has to be included in every product. Such a variable can be represented as a text replacement so that it can be replaced by one of its possible values when an individual product is instantiated.

2. Decision variables: variables representing variabilities in the decision model. Decision variables can be used in expressions (i.e., as the test value of an if statement) to select among possible variations. For example, in an auto-
marker SPL, the binary decision variable, “generate_report”, can be used to specify the inclusion or exclusion of the functional requirement.

3. Nested if-then-else: A meta-text construct that is used to select portions of the specification for inclusion or exclusion based on the value of a decision variable. We use the form: ≪ if condition then text [else text] ≫ to express the nested structure, where ≪ and ≫ denote the start and end of the construct, condition is a boolean expression over one or more of the variables, text is any legal syntactic construct, and the optional else clause has the usual semantics. For example, in an auto-marker SPL, ≪ if “use_rubric” then “rubric_location” else “define_marking_scheme” ≫ can be used to include or exclude the rubric in marking. An if construct may be nested by the inclusion of dependent if clauses in place of the text.

Like any other software systems, a SPL must evolve to fit the changing requirements of the stakeholders. A common tendency when engineers are creating reusable software is to over-generalize. It seems obvious that when we make a piece of software reusable, we should make it as generally reusable as possible. However, with over-generality comes unnecessary cost, effort, and combinatoric complexity. With SPLs, it is important to pay attention to this detrimental tendency since the entire SPL is built out of reusable core assets.

Early generation SPL methods sometimes prescribed generalizing core assets to satisfy predicted products on a 5-year horizon [35]. However, in [84], it was illustrated that it can more effective to be more agile and to generalize the core assets and product line architecture on a much shorter horizon – in their case between 3 to 6 months. The distinction between very long range scoping and very short range scoping has been termed proactive versus reactive product line scoping [34]. This is really a spectrum rather than a boolean choice in SPLs. The general advice is that we should be more proactive with analysis and architecture, while being more reactive with the development of core
2.6 Summary

In Table 2.4, we provide a tabular summary of the SPL requirements approaches surveyed in this section. For each approach, we include the major RE activities outlined in [115], including plan and elicit, model and analyze, communicate and agree, and realize and evolve.

As shown in Table 2.4, we surveyed several approaches to SPL requirements development, focusing on the literature most pertinent to the technical aspects of SPL engineering. To narrow down our discussion to the core activities in RE [115], we concentrated on the modeling effort made by the SPL and RE communities. In practice, RE for SPLs is not only a technical issue, but also a business, organizational, and process issue [35]. A more comprehensive survey of RE for SPLs would have to take the interactions of technical aspects with the other factors into account.

An interesting theme that emerged from our survey is the minimal intrusion to existing requirements modeling frameworks, such as use cases, goals and hierarchies, and requirements specifications. Even for the special purpose method – feature engineering, the trend is to rigorously define the semantics and leverage off-the-shelf tools for reasoning and debugging the models. The intention is to reuse the existing RE methods and tools as much as possible.

The survey presented in this section also identified some aspects that are as yet under-explored. Although some methods realized the existence of legacy systems, most approaches developed requirements assets proactively. Proactive approaches to SPL engineering slow its adoption by requiring substantial up-front effort and abrupt transition from current practices. For example, for Cummins to achieve its impressive SPL successes, it stopped all product deployments for six months while it rearchitected its engine...
<table>
<thead>
<tr>
<th>Approach</th>
<th>Elicit</th>
<th>Model</th>
<th>Communicate</th>
<th>Realize</th>
</tr>
</thead>
</table>
| Goal models [65, 90]                         | goals-skills-preferences 
& linguistic cases in a text corpus | goal models (esp. OR-decompositions)             | prioritization based on user skills and preferences | personalized designs and configurations     |
| Feature modeling [75, 76]                    | domain analysis                             | feature models                                  | product characteristics                         | reference architecture                      |
| Semantics of feature models [144]            | —                                           | directed acyclic graph in both syntactic and semantic domains | used by method engineers to define and compare feature languages | —                                           |
| Propositional formulas [12, 99]              | —                                           | grammars and propositional formulas             | enriched feature model debugging tools          | —                                           |
| Primitive requirements [100]                 | domain analysis & legacy systems             | domain use case diagrams                        | user-visible behavior                           | DREAM CASE tool                             |
| Extended use cases [59]                      | —                                           | use case diagrams with variation points and variants | user-relevant essential variability             | —                                           |
| Product-line requirements specification [48] | domain analysis                             | SCR tables & textual specification              | —                                               | meta-text constructs for deriving variants from the PRS |
| Definition hierarchy [85]                    | referencing standards                        | logical AND tree of quality attributes          | product prioritization                         | —                                           |
control software, organizational charts, and processes [34].

Parnas aptly summarized the dilemma faced in greenfield SPL development: we had to design the core assets for a product family at a time when we could not possibly know what members of the family would actually be built [122]. Besides, it is not usually worth building the set of core assets by itself. This set is not usually a product that anyone would ask for [121]. To resolve such a paradox, the extractive adoption model is proposed as a means of reusing existing products for the SPL’s initial baseline [34]. Core assets are no longer created from scratch, but are built by leveraging lightweight techniques without much re-engineering. The extractive approach is particularly effective for an organization that has accumulated development experiences and artifacts in a domain and wants to quickly transition from conventional to SPL engineering [151].

Our research is thus aimed at filling the SPL requirements extraction void with lightweight techniques. By lightweight, we indicate that the methods employed shall strive for increased automation and greater generality, and that our methods can be applied to perform partial analysis, without a commitment to developing a complete specification of the domain. This calls for efficient algorithms that effect a wide spectrum of domains and work on requirements artifacts in their most general form. Next, we present a framework for applying lightweight techniques to extract, model, and analyze a SPL’s requirements assets.
Chapter 3

Extracting Functional Requirements Profiles (FRPs)

This chapter discusses where, what, and how to extract, which are fundamental issues in our framework since it is the extraction results that are being modeled and analyzed. In a nutshell, we provide the following answers:

- **Where:** natural language (NL) requirements documents.
- **What:** functional requirements profiles (FRPs).
- **How:** an extraction algorithm based on information retrieval (IR) and natural language processing (NLP).

According to a recent survey [92], unconstrained NL is still the most predominant medium of requirements specifications, with 71.8% of documents provided for requirements analysis written in it. This is because text is used universally to convey information and to communicate. Despite this fact, RE research for SPLs has mainly focused on graphical representations [58, 85] and formally specified requirements [12, 48]. This leaves a large set of requirements artifacts which cannot be analyzed without expending a prohibitive effort on re-formatting them into one or the other model. We therefore choose...
NL documents to be the primary extraction source, and anticipate our textual-based technique can effect a wide spectrum of domains.

When constructing a SPL’s requirements assets, we shall follow two principles [125]: 1) Focus more on external variability (visible to customers) and less on internal variability (useful to implementers), 2) Focus more on what varies (variation point) and less on how it varies (variants). Our strategy is to tease out functional requirements profiles (FRPs) and analyze their variabilities. We define the notion of FRP to capture the domain’s action themes and to define a context for studying system qualities [107]. The FRPs in each document are identified on the basis of lexical affinities [93] and “verb–direct object” relations [147]. We present an FRP-extraction algorithm, and evaluate the results via two studies.

3.1 Motivating Example

We motivate our work with a scenario for extracting requirements assets in an auto-marker SPL. The system employed in the University of Toronto for marking programming assignments exhibited delays in turnaround time, due to the administrative burden of printing, distributing assignments to teaching assistants (TAs), and returning assignments to students. To provide feedback to students in a timely manner, a dozen teams, each consisting of 3 to 4 junior undergraduates, conducted requirements analysis and wrote software requirements specifications (SRS’s) for Web-based auto-markers in their course projects [5].

Note that these team projects were carried out separately by interviewing different sets of stakeholders. The resulting SRS’s shared certain concerns, but also had different focuses. Consolidating these results could help understand the commonalities and variations in the auto-marker domain. All 12 auto-marker SRS’s followed the IEEE-STD-830 standard in a textual form [17]. Figures 3.1a and 3.1b show the excerpts from two SRS’s
in the repository.

We are interested in culling a set of functional requirements profiles (FRPs) from these SRS’s. We define FRPs to be the action-oriented concerns [147] that bear a high information value of a document [93]. FRPs model the user-visible system functionalities, and are represented by “verb–direct object” pairs. Figure 3.1d shows a partial list of FRPs extracted from the auto-marker SRS’s.
SPL engineering considers it crucial to define a set of standard terms used in discussions about and descriptions of the domain, and makes developing a domain dictionary part of the core assets [161]. Figure 3.1c depicts a snippet of the auto-marker domain concepts: Thesaurus identifies synonym classes [140], whereas vocabulary provides the definitions of terms, acronyms, and abbreviations required to properly interpret the requirements documents [17]. These concepts are identified by domain experts. According to Figure 3.1c, we would treat “marking rubric” as a single conceptual unit, and thus determine the FRP “create marking rubric”, as indicated in Figure 3.1d.

We focus on the FRPs in building the requirements assets for a couple of reasons. First, a FRP represents the functional aspect of a feature, which is an essential characteristic of an application domain [75]. Features are distinctively identifiable abstractions that must be implemented, tested, delivered, and maintained [76]. While system qualities, such as reusability and sustainability, may become the prominent features in the long run, product functionalities remain the salient features directly observable by users.

Second, as pointed out by Bosch, starting from the functional requirements does not preclude the optimization of quality requirements during the architectural design stages [20]. In fact, we found out that having a set of concrete FRPs could effectively align quality requirements [107]. While the role of FRP in SPL engineering is further explored in Chapter 6, we now consider how to extract the FRPs from an existing requirements document.

### 3.2 Functional Requirements Extraction

The central question that we address in this section is that: Given a NL document, how can its characterizing attributes, which relate to system functionalities, be produced? We examine some single-term indexing schemes, and propose a technique for automatically generating the FRPs.
3.2.1 Single-Term Indexing

When constructing the indices for a requirements artifact, information retrieval (IR) techniques draw information from the texts rather than from a human expert. Instead of relying on a great deal of manually pre-encoded semantic information to create a knowledge base, little semantic knowledge is required and no interpretation of the document is given in most IR techniques [140]. Automatic indexing systems attempt to characterize the document rather than understand it. We prefer IR techniques in our work for reasons of cost, scalability, and domain transportability [93].

An important issue in indexing is the nature of the indices. The most usual form is a single-term index, where a term is a content identifier typically encapsulated in a word. The assumption underlying the single-term indices is stated in Zipf’s law: Given some corpus of NL utterances, the frequency of any word is inversely proportional to its rank in the frequency table [140]. John proposed some basic techniques for integrating legacy documentation assets into a SPL [70]. Use cases and system functionalities can be identified by searching the verbs with the highest frequencies of occurrence within a document. We can greatly improve the results by filtering out a list of stop words (e.g., “be” and “have”) that commonly appear in all domains [70]. The left column of Table 3.1 shows the top-ranked verbs in the AMS SRS by frequency of occurrence.

Although term frequency indicates relevance, some noise exists, mainly due to words appearing too often in a given context. In order to reduce the influence of such words it is necessary to identify the most representative terms, i.e., those containing the most information. The quantity of information of a word within a corpus is defined by its Shannon information content as:

$$INFO(w) = -\log_2(P\{w\})$$  

(3.1)

where \(P\{w\}\) is the observed probability of occurrence \(w\) in the corpus [140]. Therefore the more frequent a word is in a domain, the less information it carries. The middle column
of Table 3.1 ranks the AMS SRS’s verbs by $\text{INFO}(w)$. Most term-ranking strategies, such as tf-idf and signal-noise ratio [140], take $\text{INFO}(w)$ into account. Also note that the information value defined in (3.1) lies at the heart of many single-term IR applications in RE (e.g., [32], [63]).

### 3.2.2 Functional Requirements Profiles (FRPs)

Extracting valuable conceptual information from documentation can be done by using richer indexing units than single words. Maarek et al. used a two-word unit, called lexical affinity (LA), for profiling software libraries [93]. In linguistics, an LA between two units of language stands for a correlation of their common appearance in the utterances of the language. An LA is more restrictive than a simple co-occurrence since it necessarily relates words that are involved in a modifier-modified relation. LAs in large textual corpora have been shown to convey information on both syntactic and semantic levels, and to provide us with a powerful way of taking context into account [149].

For our purposes, we restrict the definition of LAs by observing them within a finite requirements document rather than within the whole language so as to retrieve conceptual affinities rather than purely lexical ones. One limitation of considering only a two-word unit as an LA is that domain concepts are not preserved. For example, “marking rubric” would be treated as two separate words, not as one proper term, in [93]. To address this problem, we augment our approach with a semantic component, as shown in Figure 3.1c, so that each entry in the domain vocabulary, $voc$, is maintained as one atomic conceptual unit.

Figure 3.2 outlines the algorithm for extracting domain-aware LAs. Two steps are involved in preprocessing. First, every domain concept appearing in the document is replaced by a single unit. In our implementation, for example, “marking rubric” ($e$), defined by the domain experts, is replaced by “marking_rubric” ($u_e$). Second, the re-
**Input:** a requirements document $d$, a domain vocabulary $voc$

**Output:** a list of LAs (and frequencies of occurrence) from $d$ concerning $voc$

**Preprocessing**

For each entry $e$ in $voc$

Replace every occurrence of $e$ in $d$ by $u_e$

For each word $w$ in $d$ AND $w \notin u_e$

$u_w \leftarrow$ inflectional root of $w$

**Main Procedure**

Initialize Hashtable $la_{-}freq$

For each $u \in (u_e \cup u_w)$ from the beginning to the end of $d$

Let $u_1, \ldots, u_m$ be the $m$ units immediately following $u$ in $d$

(where $m = 5$ except the end of sentence is reached earlier)

For $i = 1$ to $m$

$f \leftarrow la_{-}freq.getValue\{u, u_i\}$

($f = 0$ when $\{u, u_i\}$ has not been encountered before)

$la_{-}freq.put\{u, u_i\}, f + 1$

Return Hashtable $la_{-}freq$

---

Figure 3.2: Extracting domain-aware LAs

remaining words are stemmed [116] to permit an accurate identification of the LAs.$^1$ As a result, the word is represented by its inflectional root, i.e., the singular form for nouns and the infinitive form for verbs. The order of these two preprocessing steps is important; otherwise, “marking rubric” would be stemmed into “mark rubric”, causing the domain concept to be unrecognizable in the document.

The main procedure of the algorithm is built upon the sliding window technique [93]. The idea is to make use of an empirical observation that 98% of lexical relations relate

---

$^1$We use OpenNLP [117] to perform stemming, and discuss the evaluation in Section 3.3.
words which are separated by at most 5 words within a single sentence [96]. Therefore, most of the LAs involving a conceptual unit \( u \) can be extracted by examining the neighborhood of each occurrence of \( u \) within a span of 5 units. The window is slid throughout the document \( d \) without crossing sentence boundaries. It is worth bearing in mind that the window size of 5 conceptual units is not a parameter but a property of the English language [96]. Given that both the window size and the domain vocabulary entries are bounded by some small constants, the extraction of domain-aware LAs is linear in the number of conceptual units in the document. In the worst case where \( \text{voc} \) is not defined, the complexity of the algorithm in Figure 3.2 is \( O(n) \), where \( n \) is the number of words in \( d \).

Similar to single-term indexing, using frequency to directly determine relevance may introduce some noise. Therefore, it is necessary to select from among the LAs extracted only the most representative ones, i.e., those containing the most information. To that end, we define a measure evaluating the resolving power [93] of an LA. It is based upon the quantity of information of each of the terms involved in the LA as well as upon the frequency of appearance of this LA within the considered document. First, from the definition in (3.1), we infer the definition of information of an LA \( \{u_1, u_2\} \) as:

\[
\text{INFO}(\{u_1, u_2\}) = -\log_2(P\{u_1, u_2\})
\]

(3.2)

where \( \{u_1, u_2\} \) is a domain-aware LA retrieved while analyzing \( d \). To simplify the computation of this factor, we consider terms within the textual universe as independent variables.\(^2\) Thus we use the following formula for computing the information value of an LA:

\[
\text{INFO}(\{u_1, u_2\}) = -\log_2(P\{u_1\} \times P\{u_2\})
\]

(3.3)

\(^2\)This assumption represents only an approximation, since terms in English are definitely not independent but are distributed according to the rules of the language [93].
Then, we define the resolving power, or the quantity of information, of an LA in a given document \( d \) as:

\[
\rho((\{u_1, u_2\}, f)) = f \times \text{INFO}(\{u_1, u_2\})
\]

\[
= f \times -\log_2(P\{u_1, u_2\})
\]

\[
\approx f \times -\log_2(P\{u_1\} \times P\{u_2\}) \quad (3.4)
\]

where \((\{u_1, u_2\}, f)\) is a tuple retrieved while analyzing \( d \), meaning \( \{u_1, u_2\} \) is an LA appearing \( f \) times in \( d \), as defined by the hashtable \( \text{la_freq} \) in Figure 3.2.

The \( \rho \) score defined in (3.4) measures the information value an LA carries based on both its frequency of appearance in the text \( f \) and the quantity of information of the conceptual units involved \( (\text{INFO}(\{u_1, u_2\})) \). The LAs with high \( \rho \) scores thus effectively characterize the requirements document, but they typically include several modifier-modified relations. Consider the following sentence, taken from the EMS SRS in Figure 3.1b:

“A professor specifies mark breakdown, and records this information using a
marking rubric.”

Some of the potential LAs in this sentence are:

- of type verb-DO (direct object), e.g., “specify breakdown”, “record information”;
- of type subject-verb, e.g., “professor specify”; or
- of type noun-noun, e.g., “mark breakdown”.

We are concerned only with the verb-DO relation since our goal is to construct functional profiles. Shepherd studied the verb-DO pairs in source code and observed their denotations of action-oriented concerns [147]. More generally, an especially strong relationship exists between verbs and their themes in English. A theme is the subject matter that the action (implied by the verb) acts upon, and usually appears as a DO [25]. Thus, we define the functional requirements profiles (FRPs) of a document to be the domain-aware LAs that have a high information value (ρ) and bear a verb-DO relation. The right column of Table 3.1 lists the FRPs extracted from the AMS SRS, along with the ρ values.

One of the seminal papers in extracting useful constructs from textual requirements is Goldin and Berry [56], which introduced the tool AbstFinder. In [56], the authors treated the document as a string of characters, and when comparing two strings, they shifted one string by cutting the first character and appending it to the end in order to find repetitions. The basic assumption of this syntactic, string-processing technique is that repetition implies importance. The highly repeated (sub-)strings in a document would be identified as abstraction by AbstFinder.

Our work on FRP-extraction shares certain aspects with AbstFinder, such as both are syntactic approaches, and a good thesaurus is desirable. There are also some differences. First, FRPs use linguistic clues, not textual ones at the string level. Second, we perform semantic case analysis to facilitate variability modeling, not stopping at textual indices.
Third, our implementation is more efficient (linear in FRP-extraction versus $o(c \times N^2)$ where $N$ is number of sentences in the document and $c$ depends on the length of the sentences [56]) and makes use of off-the-shelf tools like OpenNLP. Lastly, we care about not only coverage, but also accuracy, which, in our opinion, makes FRP-extraction a little more intelligent than AbstFinder.

### 3.3 Evaluation

We carried out two studies to evaluate our FRP-extraction approach: a pilot study on the basis of the auto-marker SPL, and a case study with IBI Group’s Toronto office [66] on a set of traffic management control systems. For each study, we state the objectives and settings, and report the results and our observations.

#### 3.3.1 Pilot Study

The main objective of the pilot study is to test our approach’s feasibility. In our design, such a trial is a precursor to a larger-scale study to further assess effectiveness in an industrial setting. We use the 12 auto-marker SRS’s in the pilot trial. The auto-marker domain and the background of these SRS’s were given in Section 3.1.

Our study design is driven by such technical questions as how effective are FRPs in characterizing the user-visible system functionalities, what is the effort required to generate the FRPs, and what counts as a high resolving power? In addition to the technical inquiries, we also aim at collecting experiences and lessons learned through the interaction with the stakeholders.

We fully implemented the single-term indexing schemes and the FRP-extraction procedure. We used OpenNLP [117] for stemming and part-of-speech (POS) tagging. State-of-the-art taggers like OpenNLP have the precision of about 97%, which makes them unlikely to become an extra error source [80]. Nevertheless, we believe FRPs can tolerate
certain tagging and stemming errors because phrasal indices offer a robust way of taking context into account; testing this hypothesis is part of our future work.

For single-term evaluations, documents were indexed using stemmed verbs,\(^3\) ranked by raw term frequencies in Verb\_Freq and by term information values in Verb\_INFO. To achieve a fair comparison of FRPs, we discarded the LA containing any stop word \([47]\), and then kept only the LAs whose first units were tagged as a verb by OpenNLP \([117]\). In this study, we did not check if an LA’s second unit was a DO because the single-term indices did not have such a component. We did not perform synonym substitution either (Figure 3.1c) because Verb\_Freq and Verb\_INFO did not consider it. When incorporating these further operations, we expect to generate more effective FRPs than the ones appeared in the current pilot evaluation. The FRPs were ranked by the \(\rho\) scores, as defined in (3.4).

We adopted FAST \([161]\), one of the most mature SPL development methods, as a gold standard in evaluating the indexing schemes. FAST uses practical domain engineering for a SPL’s scope, commonality, and variability analysis. Its detailed process guidelines attracted our attention. We followed the “prepare, plan, analyze, quantify, and review” stage cycle \([161]\) in our study.

Figure 3.3 presents a flowchart that illustrates the evaluation procedure. In particular,

\(^3\)A stop list of common words must be removed from the indices \([70, 116]\). We used a domain-neutral list \([47]\) in our current implementation.
three weekly meetings were held. Each meeting lasted about one hour. Three domain experts participated in all these meetings: a TA who also acted as the moderator, an instructor, and a student. The first meeting was to establish standard terminology, the second was to ask the experts to brainstorm the domain’s action-oriented concerns. Note that the experts had not seen the automatically generated indices so far, so they were less biased toward automatic indexing. From week 2 to 3, the experts were asked to judge the relevance of the extracted indices, namely, the single verbs and the FRPs. They did the judgment separately, and agreed upon the final list during week 3’s meeting. We took the list resulted from week 3 as the gold standard to compute precision and recall.

**Effort**

As aforementioned, the algorithm for extracting domain-aware LAs is linear in the number of words contained in the document, so are the single-term indexing schemes. The cost of further obtaining verbs and FRPs is highly dependent on POS tagging and stemming. In our experience, it took OpenNLP about 5 seconds to process a 5,000-word auto-marker SRS on a PC with a 2GHz P-4 CPU and 768Mb RAM. Recent advances in requirements tagging and stemming (e.g., [116]), together with the computational efficiency of our extraction algorithm, give us confidence that our approach to functional requirements profiling can be readily scalable and extensible.

Analyzing existing requirements documents is not a replacement, but a complement, to FAST. FAST requires a considerable amount of time and effort, which is not uncommon among domain analysis methods. Our goal is to use automatic indexing to help domain experts gain insights into a SPL’s functionalities quickly and effectively.

**Effectiveness**

We assessed the quality of indices via well-known IR metrics, in comparison with the FAST’s results and the assessment of domain experts. Our FAST meetings were orga-
nized such that the experts tried to define the SPL’s functionalities before they were asked to evaluate the indices generated automatically. This helped alleviate the bias toward automatic indexing. Relevance judgments of Verb_freq, Verb_info, and FRPs were performed independently by the three domain experts, and different opinions were reconciled in the last FAST meeting.

An indexing scheme is effective if it neither produces too much irrelevant information nor misses too much relevant information. We use precision and recall [140], the most widely used measures for IR systems, to assess an indexing scheme’s effectiveness. Precision measures accuracy and is defined as the proportion of extracted information which is relevant. Recall measures coverage and is defined as the proportion of extracted relevant information to the total amount of all relevant information. Precision can also be interpreted as the probability that a retrieved (extracted) component will be relevant, and recall as the probability that a relevant component will be retrieved (extracted) [16].

Precision and recall can be defined more formally as follows: Let $A$ be the set of all relevant information, and $B$ be the set of extracted results. Let $C$ be the intersection of $A$ and $B$, i.e., the extracted parts that are relevant. Then, precision and recall are defined as:

\[
\text{Precision} = \frac{C}{B} \tag{3.5}
\]

\[
\text{Recall} = \frac{C}{A} \tag{3.6}
\]

The effectiveness comparison was performed by measuring, for the indexing schemes, precision at several levels of recall. We used the 12 auto-marker SRS’s and followed the usual procedure [140]:

1) Plotting precision-recall points for each SRS with each plot corresponding to a given recall value;
2) Extrapolating the plots to obtain precision values for recall values that were not explicitly observed; and

3) Deriving from the curves computed in stage 2) the average precision values at fixed recall intervals to obtain a single curve for the indexing scheme considered.

This procedure also equals to the following sequence of operations. First, for each document, compute precision at fixed recall values; this is achieved by looking at the top elements from the retrieval results so that the varying recall values can meet the pre-fixed values. Second, for each given recall value, compute the average precision over all the documents in the data set. Finally, connect the precision averages to extrapolate the entire curve.

We have built such curves for Verbfreq, VerbINFO, and FRPs. The curves are shown on the same axes in Figure 3.4, where 10 fixed recall values are plotted for each indexing scheme. What we are comparing here is how good the 3 indices, or the 3 linguistic clues,
Chapter 3. Extracting Functional Requirements Profiles (FRPs)

are to capture the domain’s action-oriented concerns. The best performance is reached by the scheme whose curve is closest to the area where both precision and recall are maximized – the upper right corner of the graph. The bump of the FRPs curve is due to the inability of 4 SRS’s to reach the 30% recall level or beyond; for the remaining 8 SRS’s, the average precisions keep decreasing for the recall values greater than 30%. The Verb_Freq curve slightly indicates such a fluctuation. The Verb_INFO curve, to our surprise, is so flat that the indices are indifferent. This may suggest Verb_INFO should not be applied directly; further investigating this phenomenon is part of our future work.

The results in Figure 3.4 show that for the sample SRS’s, the FRPs are better characterizations of system functionalities than the single-term indices. From Figure 3.4, it is clear that on average, FRPs have 46% better precision than Verb_Freq, and 181% than Verb_INFO. This suggests that our extraction results are much more accurate. FRPs therefore can be a good starting point for the stakeholders to understand and discuss the domain.

It turns out that, using the current analysis method, the maximum recall achieved by all three schemes is approximately the same, around 58% on the average. This implies that the extraction result of an individual system/product may not achieve a very high coverage of the domain, and further confirmed the “under-specifying principle” of assets mining [83] that the extraction results can cover the domain only partially. On one hand, as we improve our extraction method, we anticipate to achieve higher recall and precision. On the other hand, we notice that the current analysis method has its own limitations. For example, we may achieve a much higher recall if we use the union, instead of the average, to plot the recall curve at fixed precision intervals, since set union would take the results’ complementary nature into account. Therefore, in our case study described next, we employ different analysis methods to assess the results.
Although the FRPs’ effectiveness has been demonstrated, we are left with a practical problem: which FRPs shall be considered primarily. We address this question based on the $\rho$ score defined in (3.4), which measures an FRP’s information value. Setting threshold in our work is like drawing a line in the sand. Maarek et al. [93] used $\rho \geq \bar{\rho} + \sigma$ as the cutoff value for profiling software libraries, where $\bar{\rho}$ represents the mean and $\sigma$ the standard deviation of the distribution of $\rho$ within one document. In our experiment, this threshold was so selective that, for every SRS, many relevant FRPs were filtered out.

We therefore use an operational criterion – a value of $\rho$ which yields good indexing performance. Specifically, we decide the threshold based on the $F_1$-measure [140], a single value that balances both recall and precision:

$$F_1 = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

We calculate the $F_1$-measures for 3 intervals: $\rho \geq \bar{\rho} + \sigma$, $\rho \geq \bar{\rho}$, and $\rho \geq \bar{\rho} - \sigma$. Figure 3.5 shows the $F_1$-measures for 5 randomly selected SRS’s in our data set. We did not use all 12 SRS’s in this test because, in practice, it is rare that an organization faced with extractive
SPL engineering tasks has already developed more than 5 products. A common scenario is to start mining the core assets after 2 or 3 projects and keep evolving the SPL [83]. Figure 3.5 indicates that $\rho \geq \bar{\rho}$ is likely to be an optimal choice as every $F_1$-measure presents a pronounced peak centered around this threshold. However, a more decisive answer to how to make such choices requires further evaluation.

Choosing a threshold helps filter out insignificant FRPs, but the resulting FRPs can still contain irrelevant information or miss relevant information. However, this seeming drawback is really an advantage: Before the FRPs can become a SPL’s assets, they must be validated by the domain experts. From our experience with the auto-marker SPL, a 5,000-word SRS resulted in around 20 FRPs, which typically took an expert less than 10 minutes to validate. Our approach, hence, provides an efficient way to complement FAST and other domain analysis methods.

**Discussion**

We felt that, based on the auto-marker pilot trial, our FRP-extraction approach is a cost-effective method for identifying important domain elements without much human intervention. We received very positive feedbacks from meeting with the stakeholders, and also observed that establishing standard terminology could be greatly facilitated by the extraction effort. In FAST [161], establishing domain vocabulary normally spans across several meetings: the moderator prepared candidate terms; the experts attempted to agree quickly on the term’s definition; if no immediate consensus was achieved, an expert was assigned to identify alternative definitions of the term between meetings and present her recommendation to others. However, as noted in [45], in many cases, negotiating terminological differences is a waste of time. In our study, the stakeholders were able to agree quickly on the meanings of extracted FRPs within a one-hour meeting, thanks to the contextual information and the disambiguation power carried in phrasal indices [149].
Several factors can affect the validity of this pilot study. First, we did not tag the DO component within an FRP since we wanted a fair comparison to the single-term indices. Explicitly considering DO will certainly affect extraction accuracy (precision). Second, the coverage (recall) of the extraction results may not be accurately captured by the current analysis method since we pre-fixed the recall intervals. Using the analysis method that accounts for the retrieval sets’ complementarity is likely to reflect the coverage more accurately. Third, our evaluations depend on subjective judgment about the relevant FRPs to be found. Although relevance judgments are always debatable, we asked multiple experts for evaluation and reconciled the differences in a meeting. Besides, we asked the experts to define the domain’s action themes before they were asked to evaluate the automatically generated indices. This helped alleviate the bias toward automatic indexing. Lastly, the pilot trial was based on the SRS’s from students’ projects, so the findings may not be generalizable to industrial scales, which could affect the external validity of our study.

In summary, the auto-marker pilot trial successfully demonstrated the feasibility of our extraction approach. It allowed us to try out our idea in a non-trivial setting and gain firsthand experiences. At the same time, the study also shed light on some of the limitations that we want to address in further evaluations.

### 3.3.2 Case Study

**Objective and Setup**

We conducted a case study with IBI Group’s Toronto office in order to strengthen the findings reported from the auto-marker pilot trial. Our main objective is to assess the usefulness and the scope of applicability of our approach in a real-world setting. We are interested in exploring how well our approach could perform on a large industrial scale and in a domain that we are unfamiliar with.
We chose an exploratory case study [166] as the basis for the continuing empirical evaluation of our FRP-extraction approach. An exploratory case study is an in-depth exploration of one particular case (situation or subject) for the purpose of gaining depth of understanding into the issues being investigated [166]. Exploratory case studies are appropriate for preliminary studies in which it is not yet clear which phenomena are important, or how to measure these phenomena [46].

In our case, while the extractive SPL adoption model suggests some benefits [83], little has been done about the requirements so the benefits have not yet been observed empirically. Not enough is known about how exactly extracting a SPL’s requirements assets is best deployed, nor how the expected benefits arise. For these reasons, it would be premature to try to measure the cost/benefit trade-off. In this study, our intention is to explore how requirements extraction affects the organization with respect to developing a family of software-intensive systems and codifying reusable assets.

The subject in our study is a set of traffic management systems developed by IBI Group’s Toronto office [66]. IBI Group is a multi-disciplinary organization offering services in four areas of practice: urban land, facilities, transportation, and systems. As of 2008, it has over 50 offices worldwide. Toronto’s office is one of IBI’s 15 offices in Canada. We approached a project manager at IBI in early 2008 to initiate the case study. Although the group we collaborated with did not explicitly use the SPL idea to manage their products, they were interested in exploring the potential benefits offered by our research. The group was highly cooperative and generous, sharing not only their data, but also staff time and other resources.

In particular, we collected 4 SRS’s describing related but distinct traffic management systems developed by IBI. All 4 SRS’s followed the IEEE-830 standard [17] in a textual form. The average size of the main SRS’s is 5,884KB,\(^4\) which is significantly larger than

---

\(^4\)We considered only the main SRS file for each traffic management system, and ignored any appendices, exhibits, or addenda that appeared in separate files.
that of the auto-marker SRS’s (293KB). We intentionally kept the data collection at a raw level. In other words, we had little knowledge about the domain of traffic management, and we had little information about the relationships among the 4 systems under study, e.g., whether they were developed in parallel or following a particular sequential order, whether one SRS was used as a baseline for developing the others, whether the SRS’s were developed by the same group of contributors, and how similar the systems and their SRS’s are close to each other. It is our intention to address some of these issues via our method.

We held 4 meetings with IBI’s domain experts during our study. Figure 3.6 shows a flowchart that illustrates the evaluation procedure. The first meeting was to initiate the collaboration. The second meeting was to know some background information about the subject systems and their SRS’s; getting to know some terminologies was also part of the goals of this meeting. The third meeting was to ask the experts to assess the FRP-extraction results. Compared to the auto-marker study’s procedure (cf. Figure 3.3), we did not ask the expert to brainstorm the action themes before seeing the FRPs, so there was a possibility that the IBI results were biased toward the automatic indexing. However, during the third meeting, we did encourage the experts to provide any action themes that were missing from the FRP-list. In the last meeting, we presented the requirements models produced, and collected the experts’ feedbacks on both the results and our overall approach. Each of the first 3 meetings lasted about an hour, with one or

---

5 Note that each of the 12 auto-marker teams produced a single SRS file with all the appendices and figures included.
two experts participating. The last focus group meeting lasted an hour and a half; six experts attended the meeting.

Results and Discussion

The results presented in this section were obtained mainly from our third meeting with two experts. We asked the experts to assess the extracted FRPs independently. Then, the experts resolved any conflicting assessment collaboratively. We used the FRP-extraction procedure described in Section 3.2.2 in processing the 4 IBI SRS’s. Following the suggestion in [56], we removed the document’s TOC, appendices, and index before applying the automated text processing tool. Additionally, we ignored all the figures, exhibits, formulas, tables, citations, and footnotes in each SRS.

All 4 SRS’s in our data set have a section for defining vocabularies. We take advantage of this information to recognize domain concepts. Averagely speaking, the number of items defined in the SRS’s “definitions, acronyms, and abbreviations” section is 27, ranging from 9 to 72 records. Technically, not every record is relevant to FRP-extraction. For example, the definition of “user” and that of “supervisor” will not affect the domain-awareness of the FRP-extraction algorithm described in Figure 3.2, since these are single-word domain concepts. Replacing the multi-word domain concept (e.g., “traffic queue”) with a single-term unit (in our case, “traffic_queue”) requires trivial effort by using a standard word processor.

For tagging, the other preprocessing task, we continued the use of OpenNLP [117]. This time, instead of tagging only verbs, we looked at DO (direct object) as well. Since we already compared FRPs with single-term indices in the auto-marker pilot trial (cf. Section 3.3.1), our goal here is to thoroughly assess the full-fledged “verb-DO” FRPs. Similar to our experience with the auto-marker SRS’s, running OpenNLP did not present a serious performance overhead.

In Figure 3.7, we plotted the FRPs’ precision-recall curve for the IBI SRS’s by using
the procedure described in Section 3.3.1. For comparison purpose, Figure 3.7 also shows the curve from the auto-marker study (cf. Figure 3.4). What we are comparing here is how the FRPs extracted from students projects are different from those extracted from industrial-strength SRSs. The extracted IBI FRPs are more accurate, i.e., contain less false positives, than the auto-marker FRPs at 10 pre-fixed recalls shown in Figure 3.7. One reason was that DOs were explicitly considered when extracting the IBI FRPs. But in general, we felt that the effectiveness of extracted FRPs is comparable between the IBI and the auto-marker studies. Similar to the auto-marker results, the maximum recall achieved by taking the IBI FRPs with high information values ($\rho \geq \bar{\rho} + \sigma$) was around 55%. The finding once again confirmed that the results extracted from a single source could cover the domain only partially.

It is worth pointing out that, when evaluating the extraction results, a perfect 100% precision score means that every extracted FRP is indeed considered as an important action-oriented concern, and a perfect 100% recall score means that every important
action-oriented concern in the domain is identified as an extracted FRP. In order to compute the precision and recall values, an oracle is required to determine the gold standard, i.e., the sum of true positives and false negatives.

In our studies, determining the true positives was straightforward: we asked the domain expert to inspect every extracted FRP so that she could manually distinguish the FRPs that captured important action-oriented concerns (true positives) from those that did not (false positives). Even though the experts sometimes had disagreement among themselves, their conflicts were resolved quickly. In both the auto-marker and the IBI studies, it took the experts less than 10 minutes of the meeting time to agree upon all the true positives.

However, determining the false negatives was problematic in practice. Our specification of classifying domain’s important action-oriented concerns relies on the domain-aware lexical affinities that bear a “verb–direct object” relation and that carry a significant information value of the requirements document, but there might as well be the domain’s important action-oriented concerns that do not match the specification. We attempted to decide the number of false negatives by asking the experts whether there were obvious omissions from the extracted FRP-list, but this effort was far from being systematic, e.g., the experts did not review the original SRS’s or the FRPs whose information values were below the threshold. This makes determining the false negatives problematic and causes interpreting the recall score difficult.

The 55% recall of the IBI FRPs reported in Figure 3.7, for example, could indicate that each SRS contained only half of the domain’s important action themes, or that our classification specification was not very precise. We may take some actions in order to uncover the missing half. We could ask the domain engineer to do it manually, or we could match and merge the FRPs extracted from different SRS’s, as shown later in Figure 3.9. Since the gold standard must be decided by domain experts, we plan to further investigate the cost-effectiveness of our approach by conducting similar experiment as follows. First,
ask the experts to manually identify all the important action-oriented concerns in the domain, and record the effort, $E_1$. Next, run our FRP-extractor, then ask the experts to manually justify the results and find out all the missing action themes; record this effort as $E_2$. Finally, compare $E_2$ with $E_1$ to check how much our FRP-extractor saves the manual effort and whether the saving is significant.

In the auto-marker study, we employed a domain-neutral stop-word list [47] to filter out those words that commonly appear in all domains, such as “be” and “have”. A quick glance at the current IBI FRPs suggested that noise could be further reduced by discarding the indices that contain common document-formatting words. To test this hypothesis, we performed an experiment on the IBI data by adding three words to the stop-word list: “section”, “appendix”, and “following”. Figure 3.8 depicts the results, in which the average increase of precisions is $12.24\%$. Based on this promising number, we further speculate that adding to the stop-word list some (domain-specific) content words,

![Figure 3.8: Improvement made by filtering out document-formating stop words](image-url)
Chapter 3. Extracting Functional Requirements Profiles (FRPs)

e.g., “handle”, “manage”, and “information”, which are too general to represent any specific action-oriented concerns, would greatly improve the FRPs’ accuracy; however, testing this hypothesis is beyond the scope of our current work.

One problem we discovered from the auto-marker study was that the recall values may not be accurately captured using the data analysis procedure described in Section 3.3.1. We addressed this issue by thoroughly analyzing the recalls. We plotted two recall-precision curves in Figure 3.9: one for average and the other for union.\(^6\) Specifically, for each IBI SRS, we computed maximum recall at 11 pre-fixed precision values: 1.00, 0.95, 0.90, 0.85, ..., 0.50. Then, for each given precision value, we calculated the average (resp. union) recall over all SRS’s. Finally, we extrapolated the curve by connecting the 11 points in the figure. Note that the average recall curve is analogous to the average

\(^6\)We used a WordNet-based method for merging the FRPs extracted from different sources; the details will be discussed in Chapter 4.
precision curves shown in Figure 3.7 and Figure 3.8.

The results shown in Figure 3.9 confirmed our speculation that the systems’ (SRS’s) complementarity was compromised by averaging the recall values. Although the results extracted from a single source are still partial, merging the results extracted from multiple sources could reach a very satisfactory coverage: at 50% precision, IBI FRPs’ union recall reaches 92.5%, as shown in Figure 3.9. This finding is promising in that reactive SPL development [83], i.e., co-evolving the core assets and the products in a SPL, can help tackle the under-specifying problem associated with the extractive SPL adoption model.

One of our key observations was that domain concepts are extremely important to the quality of the FRPs or any (automatically generated) indices. Relying only on the document’s “definition” section is unlikely to uncover all the vocabularies used in a domain. In our case, for example, “response time” is an important concept not defined in any of the IBI SRS’s. Therefore, having domain experts validate the extracted FRPs is crucial to our approach.

In summary, the IBI case study allowed us to thoroughly assess the effectiveness, usefulness, and the scope of applicability of our FRP-extraction approach in a real-world setting. We received positive feedback from the domain experts, as well as potential areas for improvement. The results from this case study, together with those from the auto-marker pilot study, should not be seen as final definitive results, but only as an indication of what can be expected from our FRP-extraction approach. Until more SPL requirements repositories become available, it is not possible to produce statistically significant results. Producing such repositories requires a great deal of effort and is out of the scope of this thesis, but we hope that our work, as well as the work of others, will motivate this effort. In the meantime, however, our results are very promising.
Chapter 4

Modeling Requirements Assets

In the previous chapter, we showed that the extracted FRPs are capable of characterizing the domain’s action themes. However, one limitation is that the flat-list of FRPs hinders us from gaining insights into the variability properties of the SPL’s functional requirements. In this chapter, we explore how to better model the extraction results.

Modeling is an important activity in RE. A model is created for a specific purpose and contains information only needed to fulfill its purpose. Recent work suggests that requirements models, which should be palatable to users, could be applied to facilitate communication, uncover missing information, organize information, and uncover inconsistency [64].

Our focus is on modeling requirements’ variability aspect. We adopt the orthogonal variability model (OVM) [125], whose building blocks are well-thought SPL constructs: variation points, variants, dependencies, and constraints. In Section 4.1, we briefly introduce the OVM framework and relate FRPs to the OVM. We then use Fillmore’s case theory [50] (Section 4.2) to characterize each FRP’s essential semantics, and contribute a set of heuristic rules (Section 4.3) to facilitate the discovery of FRPs’ variabilities. Section 4.4 describes how we manage the models extracted from multiple sources. Section 4.5 presents an initial evaluation of our overall extraction and modeling framework.
4.1 Orthogonal Variability Model (OVM)

Documenting and managing variability is a key property that characterizes SPL engineering. The explicit definition and management of variability distinguishes SPL engineering from both single-system development and other types of reuse like software libraries [125]. Advantages of explicit variability management include:

- Decision making. Explicitly documented variability improves decision making by forcing engineers to document the rationales for introducing a certain variation point or a certain variant. The documentation of rationales can be used by customers in their choice of a certain variant, by engineers in their task of defining or binding variability, or by managers in their plans of customizing product configurations.

- Communication. Explicitly variability modeling improves communication about the variability of a SPL by providing a high-level abstraction of variable artifacts. For instance, communicating variability to customers [58] benefits from the existence of an explicit variability model. The explicit documentation of variability subjects as variation points enables customers to pinpoint the decisions to be made. The explicit documentation of variability objects as variants allows customers to consider the available options for each decision.

- Traceability. Explicitly documented variability allows for improved traceability of variability, for example between its sources and the corresponding variable artifacts. This type of link is necessary to perform application engineering effectively [159]. In addition, traceability links facilitate the implementation of changes, e.g., with respect to a variation point. Thus, the variability model of a SPL provides an entry point to navigate through all kinds of development artifacts.
In Chapter 2, we surveyed several modeling approaches. Most approaches suggest the integration of variability in existing RE models such as use cases [59] and goal models [90]. Extending traditional methods for variability modeling has some significant shortcomings. First, if variability information is spread across different models, it is almost impossible to keep the information consistent. Second, it is hard to determine, for instance, which variability information in requirements has influenced which variability information in design, realization, or test artifacts. Third, adding the variability information can make the notations over-complex. Fourth, the concepts used to define variability differ between the different kinds of existing RE models. Consequently, the variability defined in different models does not integrate well into an overall picture of the software variability. Yet, such an overall picture turns out to be essential for SPL engineering [6].

To address these problems and to make variability a first-citizen in SPL engineering, researchers have proposed to define the variability information in a distinct model. One of the most recently developed frameworks in this area is the orthogonal variability model (OVM) [125]. An OVM defines a SPL’s variability in a single view, making it orthogonal to different development phases such as requirements, design, realization, and testing. An OVM also relates the variability defined to other software artifacts such as use case models and feature models [125].

Figure 4.1 shows a sample OVM to illustrate the basic modeling constructs: variation points, variants, dependencies, and constraints. A “VP” triangle represents a variation point. It shows what can vary in the SPL. A “V” box represents a variant. It shows how the variation point (VP) varies. A mandatory variant is linked by a solid line, whereas optionals are linked by dotted lines. The alternative choice among the optionals is further annotated with an arch, along with the cardinalities specified in [min..max]. The variability constraint is given by an arrow in Figure 4.1. The constraint could be either “requires” or “excludes”. In particular, six types of constraints are defined in the original OVM notation [125]: one variant requires another variant (v_requires_v),
one variant excludes another variant \( (v\text{ excludes}_v) \), a variant requires a variation point \( (v\text{ requires}_vp) \), a variant excludes a variation point \( (v\text{ excludes}_vp) \), a variation point requires another variation point \( (vp\text{ requires}_vp) \), and a variation point excludes another variation point \( (vp\text{ excludes}_vp) \).

It is important to note that using OVM complements other software development models from the variability perspective. For example, Figure 4.2 shows the connection between an OVM and a use case diagram (adapted from [125]), in which two dotted arrows related the variants in OVM to the use cases in the use case diagram. These are “artifact dependency” arrows defined in the original OVM notation [125]. They are
different from the variability constraint arrows shown in Figure 4.1 in that they relate OVM elements to other model elements without any annotation. Discussion of artifact dependencies is out of the scope of this thesis; our goal is to integrate the FRPs into the OVM in order to help engineers build the OVM more systematically.

We regard the validated FRPs as primary constructs when building the OVM. In particular, we treat each FRP as a variation point (VP) because FRPs capture the domain’s action-oriented concerns and each product in the SPL should address these concerns in one way or another. This is in accordance with the principle in codifying a SPL’s requirements assets [125]: Focus more on external variability (visible to customers) and less on internal variability (useful to implementers). The validated FRPs characterize the user-visible system functionalities that are common across many products in the SPL, so they are sensible choices of variability subjects (external variation points in the OVM). It is argued that the external variability drives the development of the internal one [125]. Thus it is crucial to explicitly model these stakeholder concerns in the SPL’s requirements model.

It is not our intention to restrict VPs in the OVM only with the FRPs. Many other concerns can be VPs such as quality attributes, data modules, and even the users of the intended system. Nevertheless, from the external functionality point of view, we believe FRPs represent a set of core VPs that should be considered primarily in the OVM. Our work, therefore, provides some practical guidelines for generating the OVM’s elements.

4.2 Semantic Cases

Having modeled the FRPs as VPs in the OVM, we now consider systematically uncovering their variation structures. Motivated by the work on goal-based variability acquisition and analysis [90], we use Fillmore’s case theory [50] as a basis for understanding language semantics in an RE context; though, here we focus on functional requirements. The
theory analyzes the surface syntactic structure of sentences by studying the combination of cases (i.e., semantic roles like agent, object, location, etc.) which are required by a specific verb. According to Fillmore, there exists an essential set of cases that fits in the case system of every known language. Each of these universal cases addresses a particular semantic concern of the verb in a sentence, and each represents a potential semantic slot that may or must be associated with the verb. Hence, given a verb, a case frame can be defined, which is a set of semantic slots that the verb evokes. As an example, the verb “open” is necessarily associated with an objective slot (“what opens/is opened?”) but may also be associated with an agentive slot (to answer “who opens?”).

FRPs have made the DO role explicit because the verb-DO relation renders the action and its theme in English [25]. The discovery of variation structures can be driven by identifying the essential cases associated with the verb in every FRP. In this context, a case defines a variation dimension, i.e., a question whose alternative answers result in alternative refinements of the original action-oriented concern expressed by the FRP. The collection of all dimensions relevant to an FRP determines the variation structure, or the variation frame, evoked by the FRP.

Following Fillmore’s idea of defining a universal set of cases, we introduce a general set of dimensions for conceptualizing the FRP’s variation structure. The set we considered includes most of the semantic roles Fillmore originally proposed, but also draws information from recent work on variability frames for goals [90]. A high-level goal can be refined by studying the cases associated with the goal’s verb. However, a high-level goal in [90] is expressed mostly by a verb-DO pair, such as “send message” or “display record”. Such concerns will likely be recognized as FRPs in our approach. Thus, by using the auto-marker example, we consider the following variation dimensions for an FRP.

- **Agentive** defines the agent(s) whose activities will bring about the FRP’s state of affairs. Responses to this question are typically actors or combinations of actors found in the domain, including the system-to-be. For example, \{machine, TA,
instructor{Agentive “check time stamp”.

- **Objective** defines the object(s) that is affected by the FRP’s activity. Since a DO is already part of the FRP, this case concerns mainly with the *types* of DO involved. For example, “mark \{late, on-time\}Objective assignment”.

- **Locational** defines the spatial location(s) where the FRP’s activity is supposed to take place. For example, “mark assignment” \{in the lab, at home\}_Locational.

- **Temporal** defines the duration or frequency of the FRP’s activity. For example, “keep log” for \{a term, a month, a week\}_Temporal.

- **Process** refers to the instrument (P.ins) used, as well as the means (P.mea) and the manner (P.man) by which the FRP’s activity is performed. Some examples are, “access Internet” via \{Ethernet, Wireless\}_P.ins, “mark assignment” \{following marking rubric, in free form\}_P.mea, or “adjust mark” \{dramatically, subtly\}_P.man.\(^1\)

- **Conditional** defines the trigger(s) of the FRP’s action or the condition(s) under which the FRP’s function can be achieved. For example, “mark assignment” only if \{“access Internet”, “retrieve assignment”\}_Conditional.

The set is by no means an exhaustive list of grammatical features that must be associated with functional requirements descriptions, but a catalogue of categories that can help analysts understand the variation points, i.e., *what* can vary, within the FRP. Figure 4.3a shows the semantics of two sample FRPs in the auto-marker domain.\(^2\) In the semantic frame, each case defines a variation dimension for the FRP, and case’s values determine the range of that dimension. For example, only a “TA” can “mark assignment”, and the types of assignment to be marked can be “late” or “on-time”.

---

\(^1\)We will not distinguish \textit{P.ins}, \textit{P.mea}, and \textit{P.man} for the remainder of this chapter, but use the general \textit{Process} dimension instead.

\(^2\)Here we use frame-based semantics representation. Other ways to represent semantics include first-order predicate calculus, semantic network, and conceptual dependency [73].
Figure 4.3: Semantic cases and OVM

Figure 4.3b illustrates the OVM derived from the semantics shown in Figure 4.3a. As stated earlier, we map each FRP to a VP in the OVM. We then organize FRP’s variants according to their semantic roles.

It is important to note that Liaskos et al. also used Fillmore’s case theory to identify variability in goal models [90]. They took a closer look at the semantic characterization of every goal’s OR-decompositions, and pointed out the importance for distinguishing between intentional variability and background variability. Intentional variability reflects the varying means (OR-decompositions) to stakeholder goals. Background variability, also called non-intentional variability, refers to the variability that is not a result of stakeholders’ intentions, e.g., time, weather, stakeholders’ capabilities [90]. While a set of predefined intentional variability concerns should be addressed completely by the intended software, considering contextual factors simultaneously helps check whether the software system addresses all realistic background circumstances. Background variability, therefore, helps shape and constrain the stakeholder intentions. According to their
experience, Liaskos et al. showed that background variability could be effective identified by focusing on the *agentive*, *objective*, and *locational* cases [90].

Deciding which semantic cases to model is a judgement call. Depending on the task at hand, different sets of cases may be included. For example, we used agentive, objective, and conditional cases to perform cluster analysis for the FRPs [109]; Liaskos et al. emphasized the agentive, objective, and locational cases as contextual factors to be considered in goal-based variability analysis [90]. Nevertheless, the agentive case certainly plays an essential role in any kind of RE analysis, since it indicates the stakeholders involved and helps us understand their desires and needs.

### 4.3 Variability Constraints

We now discuss the intra- and inter-FRP variability issues [125]. Our purpose is to identify the variability dependencies and constraints so that FRPs can be integrated into OVM to form the SPL’s initial asset base. To that end, we present several heuristic rules for variability interdependency identification. It is important to keep in mind that variability management requires a deep understanding of the domain. Our heuristics serve as an aid to this understanding and should be treated as such. Note that this section mainly discusses building FRP-based OVMs from a single source (i.e., SRS); we will show how to merge the OVMs built from multiple sources in the next section (Section 4.4).

As shown in Figure 4.4, we extend the OVM by adding a boundary for each FRP to mark its internal variation structure. Within this structure, we organize the variants based on their variation dimensions (semantic cases). The idea of drawing a boundary is analogous to the two views — strategic dependency (SD) and strategic rationale (SR) — in the $i^*$ modeling framework [168]. From actors’ modeling point of view, A SD model

---

3The details about the cluster analysis will be discussed in Chapter 6 (Section 6.1).
Figure 4.4: Partial OVM for the auto-marker SPL

shows a zoomed out view, in which actors are related by dependency links to other actors; A SR model shows a zoomed in view, which elaborates the SD by exposing the reasoning within each actor, identifying goals, tasks, resources, softgoals, and beliefs, and their relationships (means-ends, task decompositions, softgoal contributions). In our case, we allow the user to zoom in (display) or zoom out (hide) the internal structure of any FRP to gain a comprehensive view of the OVM. In Figure 4.4, for instance, “mark assignment” is zoomed in and “retrieve assignment” is zoomed out.

The intra-FRP variability refers to the values identified along each of the case dimensions. We concentrate more on the case’s mandatory or optional property, and less on the connection between the cases within a single FRP. Note that not every FRP evokes all the cases described earlier. It is up to the domain engineer to decide which intra-FRP variabilities to model. For example, the locations where “define comment list” occurs
are not deemed essential for the auto-marker SPL, so Figure 4.4 disregards the locational case for “define comment list”.

**Heuristic 1:** “If a case is associated with only one value, then the case has one mandatory variant.”

This heuristic often applies to the agentive role to indicate the sole actor who shall perform the action. In Figure 4.4, the agent of “mark assignment” and that of “create marking rubric” instantiate this heuristic.

**Heuristic 2:** “If diverse values are identified for a case, then alternative choice(s) should be made.”

An example is “keep log” for \{a term, a month, a week\}_{Temporal}, where the values are diverse but not mutually exclusive. A \[\text{min..max}\] choice among the variants is often made in such situations, but also note that one variant (e.g., a month) can subsume another (e.g., a week). A special case of this heuristic is that the values for a case are opposite or contradictory, as in “mark \{late, on-time\}_{Objective} assignment”. In such cases, a unique \[1..1\] choice is made.

The inter-FRP variability constraint refers to the ‘requires’ or ‘excludes’ relationship between a variant and a variation point, between two variants, or between two variation points [125]. In Figure 4.4, such a constraint is represented by an annotated, dotted arrow.

**Heuristic 3:** “If FRP$_\beta$ is conditional to FRP$_\alpha$, then there exists a vp$_\text{requires}$vp constraint from FRP$_\alpha$ to FRP$_\beta$.”

This heuristic helps identify two constraints in Figure 4.4: “mark assignment” requires both “access Internet” and “retrieve assignment”. The conditional semantic case usually reveals FRPs’ interdependencies, so it seldom appears as an intra-FRP variation
Heuristic 4: “If DO_δ is a case value of FRP_φ, then there exists a v_requires_vp constraint from DO_δ to some FRP_γ such that the direct object of FRP_γ is DO_δ.”

For example, marking rubric (DO_δ) is a process value of “mark assignment” (FRP_φ), so the v_requires_vp constraint exists from marking rubric to “create marking rubric” (FRP_γ). Figure 4.4 shows this constraint, as well as a similar one on the DO comment list.

Our experience also showed that systematically identifying the variation point could uncover its variants (how it varies) that would otherwise remain hidden. For instance, it was when “mark late assignment” was identified that we noticed that on-time assignments should be marked as well. Another point to note is that even though some case dimension does not seem to be relevant at first sight, it can play an important role in deciding other cases. For example, the location to “mark assignment” does not seem to matter a lot for the auto-marker SPL initially, but due to the dependency between “mark assignment” and “access Internet”, if someone needs to “mark assignment” at the airport, the Bluetooth variant may be considered to “access Internet”.

It is worth pointing out that the above heuristic rules stem from our practice and experience of modeling the FRPs extracted from auto-marker SRS’s. They fit our intuition and lead to a trial-and-error inquiry into the SPL’s variability dependencies and constraints. Currently, only simple relationships can be identified by applying the heuristics manually. While we are keen to discover more rules and patterns, we insist they should all play a supporting role for expert opinions, because many relationships, such as [min..max] choice, ‘excludes’ constraint, and conflicts, are not likely to be inferred from text, or even from the FRPs, directly.

---

4The auto-marker pilot study was introduced in Section 3.3.1.
We conducted a preliminary evaluation of the usefulness of building the FRP-based OVM. We asked a domain expert, who did not attend the FAST meetings (cf. Section 3.3.1) and would have fresh eyes, to review the generated auto-marker OVM. She not only confirmed the OVM’s value as an asset, by which the SPL can be managed and evolved as a single and unified entity, but also spotted several constructs and relationships in the OVM to be the new insights into the domain: some examples were FRPs “highlight code segment”, “define comment list”, the process variants pre-defined comment list of “mark assignment”, and the v.excludes.v constraint from “mark late assignment” to the comment list defined by a TA.

The response was encouraging in that the OVM did not surprise the domain expert by containing obviously incorrect information. Not only that, the consolidated FRPs and their variabilities helped uncover the incompleteness and inconsistency to a certain degree. The OVM thus demonstrated its usefulness by guiding people in keeping an eye on the important issues. We present a more thorough evaluation of the IBI case study in Section 4.5.

### 4.4 Merging FRP-based OVMs

Having modeled the FRPs extracted from a single source, we must compare the FRPs and the OVMs built from multiple sources. We view this problem as an instance of model matching and merging [104]. Since special attention must be paid to SPL variability, we first discuss the annotation of the mandatory and optional properties in the OVM.

When analyzing a natural language SRS, mandatory and optional attributes can often be inferred from the categories of words used in the document. Several quality indicators were defined in [163] to assess individual specification statements and the vocabulary used to state the requirements. Of particular interest are the metrics developed for measuring *imperatives* and *options*. Imperatives, which we use to detect mandatory requirements,
are those words and phrases that command something must be provided. The use of words such as “shall” and “must” is a strong indicator for a forceful statement of a requirement. In particular, “shall” is usually used to dictate the provision of a functional capability [163], and is of special value in helping determine the mandatory-ness.

Options, on the other hand, is the category of words that give the engineer latitude in satisfying the specification statements that contain them [163]. The words that most frequently appear in this category include “can”, “may”, and “optionally”, which we use to help decide the optionality. We recommend judicious use of these indicators since neither imperatives nor options are definitive. In the auto-marker SPL, for example, “request remarking” is actually a mandatory FRP despite its companionship with the optional indicator “can” in the AMS SRS (Figure 3.1a).

The annotation of mandatory or optional properties can be applied to not only the variants but also the variation points (FRPs) in the OVM. In the original OVM proposal, variation points (VPs) are mandatory to all the products since they are the variability subjects and every product in the SPL must address them. However, since we treat FRPs as VPs, we feel that it is necessary to have the ability to distinguish mandatory and optional VPs, that is, to distinguish the system functionalities that are common to all the products and those that are unique to some products in the SPL. This extension can be easily implemented by using the current OVM notations [125]: solid line to connect mandatory VPs and dotted line to connect optional VPs (cf. Figure 4.1). In Figure 4.5, for instance, “calculate travel time” is linked by a dotted line, indicating that this FRP (VP) is optional to some of the products in the SPL.

Since we regard the problem of integrating the FRP-based OVMs as an instance of model matching and merging [104], we need a matcher and a merger in our framework. A matcher finds correspondences between a given set of the model elements, and a merger combines the OVM set to form the SPL’s initial asset base.

In the literature, specification similarities in OVM can be rendered in different di-
mensions, such as analogical [95], conceptual [139], structural [21], and behavioral [169]. We currently match the OVMs at the variation points (FRPs) level, i.e., relating similar FRPs (extracted from different sources) to each other. Specifically, we make use of two clues to build the correspondence: the domain-specific thesaurus and the domain-neutral WordNet [164].\(^5\) The domain-specific thesaurus defines synonym classes (cf. Figure 3.1c). We employ such a thesaurus in two ways: the terms used in expressing the FRP and the terms that used in expressing the variants within an FRP’s variation structure. A practical strategy we currently adopt is to preprocess the SRS’s by replacing all synonym occurrences by a designated term selected by a domain expert. For example, both “AMS” (Assignment Marking System) and “EMS” (Electronic Marking System) would be replaced by the designated term “auto-marker”. We then use the freely available WordNet::Similarity package [123] to compute the similarity between the OVM elements’ name labels based on their linguistic correlations. The domain expert is involved in determining the similarity threshold and validating the matching results.

\(^5\)WordNet is a lexical database for the English language. It groups English words into sets of synonyms called synsets, provides short, general definitions, and records the various semantic relations between these synonym sets.
Our merger takes the given set of FRP-based OVMs, together with the corresponding relationships found by the matcher, and produces a merge that preserves the mandatory and optional properties of the input FRPs. Our merge operator acts as conjunction [171], and also satisfies the desired algebraic properties defined in [21], such as idempotency, commutativity, associativity, etc. A novel aspect of our approach is to employ a knowledge ordering during merging: optionality takes precedence over mandatory-ness. That is, the merged FRP or its variant is mandatory only if it is defined as mandatory in all individual SRS’s. Otherwise, the OVM element is optional in the SPL. This merging process fits our intuition and is also amenable to automation. The merged OVM forms an initial asset base, by which we can manage and evolve the software family as a single, unified entity. Figure 4.5 illustrates the merged OVM for the IBI’s “calculate travel time” FRP, in which the contributing product to the optional variant is annotated. For example, product A (V[A]) uses the “ITTMS TMS” algorithm to calculate travel time, whereas product B (V[B]) uses the “CTT function”.

In summary, the integration of FRP-based OVMs is handled by our generic merging framework [21]. One of the advantages of using FRPs in the merge process is that we can compare different SRS’s by inspecting their respective profiles (FRPs) so as to gain insights into the relationships between the products in the SPL. When comparing a pair of SRS’s provided by IBI, SRS_A and SRS_B, we use the following measure:

\[
\delta_{AB} = \frac{\text{FRP}_A \cap \text{FRP}_B}{\text{FRP}_A \cup \text{FRP}_B}
\] (4.1)

where FRP_A and FRP_B are the sets of validated FRPs for product A and B respectively.

### 4.5 Evaluation

In order to show that our approach is capable of extracting and codifying the essential SPL assets, the FRPs and the FRP-based OVMs must be reviewed by experts and ad-
justed according to domain knowledge. In Section 4.3, we described an initial evaluation of the usefulness of the generated auto-marker OVM. The positive response from the domain expert urged us to conduct a more thorough evaluation via the IBI case study.\footnote{The general background of the IBI case study was introduced in Section 3.3.2.}

We conducted a focus group session in our fourth meeting with the IBI experts. A focus group is a qualitative research method that collects data through group interaction on a topic determined by the researcher \[101\]. Focus groups are thus carefully planned discussions, designed to obtain personal perceptions of the group members on a defined area of research interest. There are typically between 3 and 12 participants in a focus group and the discussion is guided and facilitated by a moderator-researcher, who follows a predefined questioning structure so that the discussion stays focused. In \[81\], the authors discussed the use of focus group in empirical software engineering.

We devised a questionnaire with eight free-form questions to guide the focus group design.\footnote{The questionnaire is provided in Appendix A of this thesis.} The goal was to assess our modeling effort and the overall approach by obtaining feedbacks and insights from the group discussions. During the focus group’s execution, the moderator-researcher presented to the participants the FRPs extracted, the product similarities measured by overlapping FRPs (cf. (4.1)), and the OVMs built from the SRS’s. We used the presentation and the questionnaire to stimulate the discussion. At predefined breakpoints, the moderator asked the participants to respond to certain questions from the questionnaire, and allowed the participants to talk about the topics among themselves. Six IBI experts participated in the session. The focus group lasted an hour and a half. We collected three completed questionnaires at the end, all of which were anonymous.

Obviously, the number of sample users of our framework is not representative of the community of domain engineers or requirements modelers, so any quantitative data analysis will lack statistical significance and credibility. However, qualitative data anal-
Table 4.1: Top 5 FRPs extracted

<table>
<thead>
<tr>
<th>Product A</th>
<th>Product B</th>
<th>Product C</th>
<th>Product D</th>
</tr>
</thead>
<tbody>
<tr>
<td>generate response</td>
<td>detect queue</td>
<td>generate response</td>
<td>configure MassTERs</td>
</tr>
<tr>
<td>display state</td>
<td>generate response</td>
<td>detect queue</td>
<td>generate response</td>
</tr>
<tr>
<td>determine subscriber</td>
<td>display state</td>
<td>generate broadcast</td>
<td>detect incident</td>
</tr>
<tr>
<td>detect incident</td>
<td>generate broadcast</td>
<td>display state</td>
<td>implement response</td>
</tr>
<tr>
<td>send report</td>
<td>use CCTV</td>
<td>display message</td>
<td>specify recipient</td>
</tr>
</tbody>
</table>

ysis [133] can give an initial reaction to how our approach is considered and perceived by the targeted users. Qualitative data are records of observation or interaction that are complex and contextualized, and they are not easily reduced immediately, or sometimes ever, to numbers. Qualitative research seeks to make sense of the way themes and meanings emerged and patterned in the data records built up from observations, interviews, surveys and questionnaires, and other research media [133]. In our evaluation, we used coding (relating answer sections to proper subject matters under testing) and categorizing (classifying answers to be positive or negative) [133] when analyzing the collected data. In the remainder of this section, the direct quotes from the respondents are shown in italic and cited in double quotation marks (" ").

**Question 1 (Scope):** Do you think the model elements (FRPs and semantic cases) capture important domain elements? Are the results surprising? Insightful?

**Subject matter under testing:** Scope of the assets.

This question is a continuing test of the quality of extracted FRPs, since we already assessed the effectiveness of FRP-extraction and presented the results in Section 3.3.2. In the meeting, we showed the top-5 FRPs extracted from each product’s SRS. Table 4.1 shows the results.\(^8\) The selection of 5 FRPs was deliberate since we did not want to

\(^8\)To honor confidentiality agreements, we will use the pseudonyms A, B, C, and D to refer to the four IBI products. In the IBI meeting’s presentation, we used the following color scheme to refer to the different products: blue for A, brown for B, red for C, and magenta for D. While we kept this color scheme in presenting the results in this section, we also explicitly annotated the variants by using the
overwhelm or bore the domain experts by tens of FRPs for each product. After all, the relevance of extracted FRPs was judged by one IBI expert in the third meeting. While we threatened the responses to this particular question, we achieved the objective of warming up the experts and keeping them engaged.

The answers to this question were positive in that “not that surprising” was the representative response. This confirmed that the FRPs (at least the top-ranked ones) indeed captured the domain’s important action themes. However, one expert pointed out that “event management seems to be missing”. A closer look at the complete result sets indicated that the FRPs related to event management, such as “plan event”, “schedule event”, and “manage event”, appeared around the fifteenth among the FRPs of C’s and D’s. Should we include more FRPs in the presentation, we would cover the important domain elements more completely.

**Question 2 (Products’ Similarity):** Do you find using the FRP-differences for assessing products’ similarity is sensible? Promising?

**Subject matter under testing:** Usefulness of using FRPs to compare the products.

We used (4.1) to calculate the similarity between the four products. The results were given in Table 4.2. We drew the conclusion that A, B, and C were close to each other, whereas D was different from the other three. Although one may argue that the similarity measure was rather crude, the experts’ responses were very positive: “was product pseudonyms when necessary.
pretty accurate in IBI’s case” and “interesting and promising” were among the answers. Given that we, as researchers, knew little about the relationships among the 4 products, the findings were especially encouraging. Further investigation could offer finer-grained similarity measures so as to select the proper product(s) to maximize reuse.

**Question 3 (OVMs):** Do you think the commonality, variability, and dependency captured in the OVMs are accurate? Insightful?

**Subject matter under testing:** Quality of OVMs.

We presented three OVMs during our presentation: “calculate travel time”, “detect queue”, and “generate response plan”, which are shown in Figure 4.5, Figure 4.6, and Figure 4.7 respectively. As pointed out earlier, we modeled “calculate travel time” as an optional FRP since this feature was pending in product B. Note that, in Figure 4.6, “monitor queue” is zoomed out because it is not the focus of the current view.

The answers to this question were quite positive, in which one expert responded, “fairly accurate given limited knowledge of products”. Furthermore, some subtleties appeared in the OVMs attracted much attention and turned out to be very insightful. For
instance, in Figure 4.6, the variant-excludes-variant constraint from “incident Objective” to “Multi-threshold Process” would be a constraint across all 4 products; however, we found that only product D explicitly stated it. Such a finding helped provoke useful discussions among the stakeholders. As one response stated, “known beforehand but a clear way to present”.

Questions 4 – 8 in the questionnaire were designed to assess our overall approach to extracting and modeling the requirements assets. The design is based on diffusion
theory [136], which examines the rate and the motivations of adoption of a technological innovation by a group of potential users. Such an approach may also be fruitful for the evaluation of a novel conceptual framework (such as a design or requirements method), by assessing whether it is appreciated by a community of stakeholders [74].

The diffusion theory defines five perceived quality attributes of an innovative product. Triability is the degree by which the product can be tried on a limited basis before adoption. Observability refers to the observable results deriving from the use of the new product. Relative advantage is the perception of how much better the innovation is than the competing solutions currently adopted. Complexity refers to the fact that the innovative product should not be overly complex to understand and to use. Compatibility measures how the innovation is perceived as compatible and consistent with existing practices shared among the community of users [136]. We felt that a qualitative evaluation of our approach based on these attributes would be appropriate, since we were still in the process of theory exploration and formulation, trying to reflectively learn something during the evaluation exercise rather than definitely test already known hypotheses.

**Question 4:** According to your experience, do you think that this approach (FRPs + OVMs) provides sufficient constructs and guidelines to be tested on a limited basis before adoption?

**Subject matter under testing:** Triability.

The answers were somewhat positive: one responded “yes” and the other two suggested that additional case studies would be needed.

**Question 5:** Do you see preliminary observable results from the application of the proposed approach to extracting and modeling a domain’s requirements assets?

**Subject matter under testing:** Observability.

The answers were very positive: all the experts responded “yes”. One further con-
firmed that “such a tool would be valuable in a company like IBI”.

**Question 6:** Compared to relevant techniques you are aware of, do you think that the adoption of the proposed approach can better help you improve the quality of requirements engineering (elicitation, analysis, documentation, etc.) for a software product line?

**Subject matter under testing:** Relative advantage.

The answers were inconclusive: one said “not sure, but it shows promise”, one said “you still have to apply considerable application domain knowledge to interpret the results”, and the third expert misunderstood the question by saying that “Yes. I think quality SRS’s would have made the FRPs more accurate”.

**Question 7:** Do you think that the proposed approach is overly complex to be understood and used?

**Subject matter under testing:** Complexity.

The answers were positive with all respondents replying “no”.

**Question 8:** Do you perceive the proposed approach to be compatible and consistent with the existing practices, values, standards, and technologies shared in your organization?

**Subject matter under testing:** Compatibility.

The answers were diverged. One expert supported compatibility: “yes”; another opposed: “No. This is radically different”. The third expert provided a more comprehensive, yet compromised, opinion: “We don’t typically spend time analyzing this type of info as we don’t do R&D. We are much more project-based. That being said, there was value looking at this, especially with new eyes”.

In summary, we used the SCV (scope, commonality, and variability) criteria [37] to evaluate the quality of the OVMs, and the attributes defined in diffusion theory [136]
to evaluate whether our overall approach can be spread more widely. We concluded the results were positive and encouraging, but would like to improve future study designs by addressing some of the threats to our current exploratory case study’s validity [166].

- Construct validity concerns about establishing correct operational measures for the concepts being studied. In our case, one expert misinterpreted the construct “relative advantage”, which dictates further effort to revise some of the descriptions and questions.

- Internal validity concerns about establishing a causal relationship, whereby certain conditions are shown to lead to other conditions, as distinguished from spurious relationships. While we did not set up explicit hypotheses to test, we were able to generate some competing hypotheses for further testing. For example, when evaluating “products’ similarity” (question 2), one domain expert pointed out that the similarity between the SRS’s “may have more to do with SRS authors than functionality”.

- External validity concerns about establishing the domain to which a study’s findings can be generalized. We felt that the threat to external validity was not serious in that the experts were impressed by the accuracy of the results, given that little application domain knowledge was involved in producing them.

- Reliability concerns about demonstrating that the operations of a study – such as the data collection procedures – can be repeated, with the same results. We expect that replications of our study should offer results similar to ours. Of course, the characteristics of the organization and the software systems under study will differ from our reports, but the underlying trends should remain unchanged.
Chapter 5

Analyzing Quality Requirements

Functional requirements describe what the system shall do. Quality requirements describe how well the system shall do it. We often hear such desired qualities as reliability, efficiency, and usability. The term ‘quality requirements’ is really an umbrella term [15]. In the literature, they are also known as nonfunctional requirements, softgoals, and quality attributes [115]. Note that this chosen terminology shall not be confused with high-quality requirements.

In this chapter, we investigate both functional and quality requirements via concept analysis [54]. Our goal is to efficiently capture and evolve a SPL’s assets so as to gain insights into requirements modularity and interactions. To that end, we first set the context by leveraging FRPs and the SEI’s quality attribute scenarios [10]. By analyzing the relation in context, the interplay among requirements is identified and arranged in a so-called concept lattice. We then formulate a number of problems, with an emphasis on quality requirements, that SPL RE should address, and present our solutions according to the concept lattice. In particular, we locate quality-specific functional units, detect interferences, update the concept hierarchy incrementally, and analyze the change impact.

The contributions of the work presented in this chapter lie in the novel use of concept lattice for investigating the relationships among sets of requirements. Our approach
complements traditional methods by automating some laborious RE tasks in the presence of an evolving SPL. We evaluate our approach on a case study of a mobile game SPL. We not only analyze the requirements artifacts, but also discuss the requirements practice in the organization. The study shows that concept lattice offers remarkable insights into modularity and abstractions, and that lightweight analysis can be integrated into an organization’s current RE practice to facilitate the development of reusable assets.

5.1 Preliminaries

5.1.1 Aspect-Oriented RE and Quality Attribute Scenarios

Requirements abstractions are decomposed functionally in most contemporary SPL approaches like FODA (feature-oriented domain analysis) [75] and PRS (product-line requirements specification) [48], since system functionality represents the very noticeable aspect of a feature [75]. The situation becomes messier when quality requirements, like safety and portability, are considered: they impact and crosscut multiple functional modules. The work on definition hierarchies [85] used quality requirements as the primary decomposition criteria because quality concerns drive architectural design. However, the resulting hierarchies are rather isolated with each rooted in a distinct concern. This presents an obstacle to quality trade-off analysis, and also causes SPL features to repeat themselves across different hierarchies.

Aspect-oriented requirements engineering (RE) aims to overcome the deficiencies of traditional abstractions by developing richer notions of modularity for requirements [131]. The role of aspects in modeling SPL variabilities was examined in [91]. We view the role of aspects in SPL RE, rather broadly, as a means of enhancing modularity, detecting conflicts, analyzing trade-offs, and supporting evolution. Our view is suited for the extractive and reactive SPL approaches [83]. The extractive approach reuses existing artifacts for the SPL’s initial baseline. The reactive approach is like the spiral or extreme
programming approach to conventional software: it embraces change and makes assets and products co-evolve. We argue that, in both approaches, aspects play an essential role in understanding a SPL’s requirements dependencies and interactions.

Quality requirements not only describe how well system functions are accomplished, but also represent global concerns that are natural to be implemented as aspects [79]. In addition, they guide the selection between various design options, so they are a SPL’s architectural drivers (also called architecturally significant requirements). Real life SPL development often emphasizes on satisfying a few significant requirements because any change in them is very likely to affect the entire architecture. Therefore, quality requirements, especially the architectural drivers, should be used as the top most elements in analysis and design [85]. The benefits of getting quality requirements “right” are substantive in SPL engineering [20], but applying them raises many practical questions. First of all, quality requirements tend to exhibit trade-offs that must be carefully negotiated and resolved. For example, modifiability affects performance, scalability affects reliability, and everything affects cost. Engineers must find an architectural solution that balances these competing needs. Moreover, quality requirements are difficult to measure because they tend to be achieved within acceptable limits rather than absolutely. The fact that globally concerned qualities cut across many subsystems [79] also makes tracking such concerns a difficult task.

Before starting any analysis work, we must proactively elicit quality requirements. A fundamental challenge of communicating quality requirements is that stakeholders do not use consistent terminologies [107], even though several standards (e.g., ISO/IEC 25030) exist [15]. On one hand, a standard hardly covers all stakeholder concerns in every possible domain. On the other hand, stakeholders may use quality terms in idiosyncratic ways, resulting in ambiguous and conflicting requirements descriptions. Our previous work showed that concrete functional requirements profiles (FRPs) could form a common ground for tackling the terminological mismatches between quality requirements [107].
The validated FRPs become assets of the SPL because they represent the domain’s action themes and every product in the product line shall address them in one way or another.

Although the FRPs help identify the functional units, there is still a gap in relating them to quality requirements, e.g., a great deal of domain expertise was spent in uncovering their relationships [107]. To bridge the gap, we take advantage of the SEI’s quality attribute scenarios [10]. Scenarios – brief narratives of expected or anticipated system uses from both user and developer views – provide a look at how the system satisfies quality attributes in various use contexts. The main difference between these scenarios and the use case scenarios or user stories is that they must specify quality attribute information. In another word, every scenario \(^1\) provides an operational definition for some quality attributes [10]. For example, it is meaningless to say that a system is modifiable. Every system is modifiable with respect to one set of changes and not modifiable with respect to another. It is more meaningful to cast the requirement as a scenario, such as:

“A developer wishes to add a searching input field and button to the UI code, as well as to resize the toolbar icons; modifications shall be made with no side effect in three hours; the resulting system addresses items 5 and 13 in version 1.0.2’s bug report so usability is expected to increase.”

The scenarios make quality requirements measurable, and also help resolve terminological ambiguities by capturing the stakeholders’ precise concerns. This let us supplement the terms different stakeholders use with a specification that is independent of any standard or taxonomy. Since scenarios are an asset that can be reused in analyzing a family of related systems [77], our work uses them to investigate the relationship and modularity of a SPL’s requirements.

\(^1\)For the rest of the paper, “scenario” refers to the quality attribute scenario, unless otherwise noted.
5.1.2 Concept Analysis

Concept analysis, or formal concept analysis (FCA), is a mathematical technique for analyzing binary relations. The mathematical foundation of concept analysis was laid by Birkhoff [14] in 1940. For more detailed information on FCA, we refer to [54], where the mathematical foundation is explored.

FCA deals with a relation \( I \subseteq O \times A \) between a set of objects \( O \) and a set of attributes \( A \). The tuple \( C = (O, A, I) \) is called a formal context. For a set of objects \( O \subseteq O \), the set of common attributes \( \sigma(O) \) is defined as:

\[
\sigma(O) = \{ a \in A \mid (o, a) \in I \text{ for all } o \in O \}. \quad (5.1)
\]

Analogously, the set of common objects \( \tau(A) \) for a set of attributes \( A \subseteq A \) is defined as:

\[
\tau(A) = \{ o \in O \mid (o, a) \in I \text{ for all } a \in A \}. \quad (5.2)
\]

A formal context can be represented by a relation table, where columns hold the objects and the rows hold the attributes. An object \( o_i \) and attribute \( a_j \) are in the relation \( I \) if and only if the cell at column \( i \) and row \( j \) is marked by “\( \times \)”. As an example related to the media shop, a binary relation between a set of objects \( \{CD, MAGAZINE, NEWSPAPER, VIDEOTAPE, BOOK\} \):

<table>
<thead>
<tr>
<th>MEDIA SHOP</th>
<th>free-distribution</th>
<th>timely</th>
<th>paper</th>
<th>sound</th>
</tr>
</thead>
<tbody>
<tr>
<td>CD</td>
<td>( \times )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAGAZINE</td>
<td>( \times )</td>
<td>( \times )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NEWSPAPER</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
<td></td>
</tr>
<tr>
<td>VIDEOTAPE</td>
<td></td>
<td></td>
<td></td>
<td>( \times )</td>
</tr>
<tr>
<td>BOOK</td>
<td></td>
<td></td>
<td></td>
<td>( \times )</td>
</tr>
</tbody>
</table>

(a) Formal context.

(b) Concepts for the formal context.

<table>
<thead>
<tr>
<th>( T )</th>
<th>(({CD, MAGAZINE, NEWSPAPER, VIDEOTAPE, BOOK}, \Phi))</th>
</tr>
</thead>
<tbody>
<tr>
<td>( c_1 )</td>
<td>(({CD, VIDEOTAPE}, {\text{sound}}))</td>
</tr>
<tr>
<td>( c_2 )</td>
<td>(({CD, NEWSPAPER}, {\text{free-distribution}}))</td>
</tr>
<tr>
<td>( c_3 )</td>
<td>(({MAGAZINE, NEWSPAPER, BOOK}, {\text{paper}}))</td>
</tr>
<tr>
<td>( c_4 )</td>
<td>(({MAGAZINE, NEWSPAPER}, {\text{timely, paper}}))</td>
</tr>
<tr>
<td>( c_5 )</td>
<td>(({NEWSPAPER}, {\text{free-distribution, timely, paper}}))</td>
</tr>
<tr>
<td>( c_6 )</td>
<td>(({CD}, {\text{free-distribution, sound}}))</td>
</tr>
<tr>
<td>( \bot )</td>
<td>((\Phi, {\text{free-distribution, timely, paper, sound}}))</td>
</tr>
</tbody>
</table>

Figure 5.1: Binary relation
VIDEOTAPE, BOOK} and a set of attributes \{free-distribution, timely, paper, sound\} is shown in Figure 5.1a. For that formal context, we have:

\[
\sigma(\{\text{CD}\}) = \{\text{free-distribution, sound}\},
\]
\[
\tau(\{\text{timely, paper}\}) = \{\text{MAGAZINE, NEWSPAPER}\}.
\]

A tuple \( c = (O, A) \) is called a concept if and only if \( A = \sigma(O) \) and \( O = \tau(A) \), i.e., all objects in \( c \) share all attributes in \( c \). For a concept \( c = (O, A) \), \( O \) is called the extent of \( c \), denoted by \( \text{extent}(c) \), and \( A \) is called the intent of \( c \), denoted by \( \text{intent}(c) \). Informally speaking, a concept corresponds to a maximal rectangle of filled table cells modulo row and column permutations. In Figure 5.1b, all concepts for the relation in Figure 5.1a are listed.

The set of all concepts of a given formal context forms a partial order via the superconcept-subconcept ordering \( \leq \):

\[
(O_1, A_1) \leq (O_2, A_2) \iff O_1 \subseteq O_2, \quad (5.3)
\]

or, dually, with

\[
(O_1, A_1) \leq (O_2, A_2) \iff A_1 \supseteq A_2. \quad (5.4)
\]

Note that (3) and (4) imply each other by definition. If we have \( c_1 \leq c_2 \), then \( c_1 \) is called a subconcept of \( c_2 \) and \( c_2 \) is a superconcept of \( c_1 \). For instance, in Figure 5.1b, we have \( c_5 \leq c_3 \).

The set \( L \) of all concepts of a given formal context and the partial order \( \leq \) form a complete lattice, called concept lattice:

\[
L(C) = \{(O, A) \in 2^O \times 2^A \mid A = \sigma(O) \text{ and } O = \tau(A)\}. \quad (5.5)
\]
The infimum ($\cap$) of two concepts in this lattice is computed by intersecting their extents as follows:

$$ (O_1, A_1) \cap (O_2, A_2) = (O_1 \cap O_2, \sigma(O_1 \cap O_2)). $$  \hspace{1cm} (5.6)

The infimum describes a set of common attributes of two sets of objects. Similarly, the supremum ($\sqcup$) is determined by intersecting the intents:

$$ (O_1, A_1) \sqcup (O_2, A_2) = (\tau(A_1 \cup A_2), A_1 \cup A_2). $$  \hspace{1cm} (5.7)

The supremum yields the set of common objects, which share all attributes in the intersection of two sets of attributes.

The concept lattice for the formal context in Figure 5.1a can be depicted as a directed acyclic graph whose nodes represent the concepts and whose edges denote the superconcept-subconcept relation $\leq$ as shown in Figure 5.2. By convention, the edges are not provided with arrowheads; instead, the superconcept always appears above its subconcepts. In the sparse concept lattice, a node $n$ is marked with an attribute $a$ if the
node is the most general concept that has $a$ in its attribute set. Similarly, a node $n$ is marked with an object $o$ if the node is the most special concept with $o$ in its object set. Attribute sets are shown just above each node, whereas object sets are shown below the node. In general, we can use the following simple rule to help read the diagram:

An object $o$ has an attribute $a$ if and only if there is an upwards leading path from the node named by $o$ to the node named by $a$,

or, dually, with

An attribute $a$ is possessed by an object $o$ if and only if there is a downwards leading path from the node named by $a$ to the node named by $o$.

Hence, we recognize from the line diagram in Figure 5.2 that the node “MAGAZINE” has exactly the attributes “timely” and “paper”, and the node “timely” has exactly the objects “MAGAZINE” and “NEWSPAPER”. As a consequence of the reading rule, we can easily read from the line diagram the extent and the intent of each concept by collecting all objects below respectively all attributes above the node of the given concept. Hence, the object concept “MAGAZINE” has the extent “MAGAZINE” and “NEWSPAPER” and the intent “timely” and “paper”. The extent of the top concept ($\top$) is always the set of all objects, and the intent of the bottom concept ($\bot$) is always the set of all attributes. While in the context of Figure 5.1a, the intent of the most general concept ($\top$) does not contain any attribute. In other contexts, it may occur that the intent of $\top$ is not empty. For example, if we add to the given context the attribute “media” with crosses in each row in Figure 5.1a, then the top concept would be the attribute concept of “media” and the intent of $\top$ would contain just the attribute “media”.
5.1.3 Running Example

We use the media shop [106] in the e-commerce domain to illustrate our approach; a fuller case study is presented in Section 5.4. Media shop is a store selling different kinds of media items such as books, magazines, audio CDs, and videotapes. A family of media shops can vary in the products to sell, the payment methods, the accepted currencies, and the like. In order to make requirements analysis more straightforward, we develop the following quality attribute scenarios:

Sce\(_1\): A user is able to navigate the media shop by categories, search the product, customize her own toolbox, and select her native language (e.g., English, Spanish, Japanese) for displaying the product and price information; usability is enhanced due to navigation aids and customization capabilities, but maintenance is likely to experience extra overhead.

Sce\(_2\): An administrator wants to monitor product quantities while navigating the shop; he also wishes to customize toolbox for automatically generating the reports; this makes the system easier for him to use and access.

Sce\(_3\): To support the accessibility requirements, the developer must create admin account(s) and provide mechanisms for navigating the shop, searching the product, and monitoring the quantities; these features will make further maintenance troublesome.

Several points are worth mentioning. First, scenarios are descriptions of tasks associated with stakeholder roles [77], so we characterize system uses from various views. Second, although some scenario formatting template (e.g., stimulus, artifact, response) is proposed [10], we tend not to impose how a scenario is structured, but support narratives in the general sense [77]. Third, each scenario necessarily contains some action-oriented concerns, and by definition, each scenario provides quality attribute information. Table 5.1 lists the FRPs and the quality requirements considered in the scenarios. Finally,
5.2 Extractive Requirements Analysis — Symmetric View of Aspects

The symmetric view does not support the notion of base where aspects are weaved. Rather, every aspect represents a concern in its own dimension, and is projected to other dimensions according to its impacts on other concerns [153]. In dealing with the requirements extracted from existing systems, every functional unit or quality attribute represents a concern in its own right. Thus, we adopt the symmetric view to help understand interactions and trade-offs between requirements.

Context is critical for concerns, especially quality ones, to be precisely specified in the multi-dimensional space. There are no simple (scalar) “universal” measurements for qual-

---

<table>
<thead>
<tr>
<th>Functional Requirements Profiles</th>
<th>Quality Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>FRP₁: navigate shop</td>
<td>Q₁: +U (positively contribute to usability)</td>
</tr>
<tr>
<td>FRP₂: search product</td>
<td></td>
</tr>
<tr>
<td>FRP₃: customize toolbox</td>
<td>Q₂: +A (positively contribute to accessibility)</td>
</tr>
<tr>
<td>FRP₄: select language</td>
<td></td>
</tr>
<tr>
<td>FRP₅: monitor quantity</td>
<td>Q₃: −M (negatively contribute to maintainability)</td>
</tr>
<tr>
<td>FRP₆: generate report</td>
<td></td>
</tr>
<tr>
<td>FRP₇: create account</td>
<td></td>
</tr>
</tbody>
</table>

scenarios are inherently incomplete so their use may be limited in supporting some RE activities like elicitation. However, they play an important role in our work for analyzing the interplay of functional and quality requirements. In fact, the under-specifying principle [83] suggests that attempting to build a complete set of requirements is counterproductive; the assets will be enriched when abstractions emerge from the SPL’s evolution.
Table 5.2: Crosscutting relations

<table>
<thead>
<tr>
<th></th>
<th>FRP</th>
<th></th>
<th>Q</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 2 3 4 5 6 7</td>
<td></td>
<td>1 2 3</td>
<td></td>
</tr>
<tr>
<td>Sce₁</td>
<td>×  ×  ×  ×</td>
<td>×</td>
<td>×</td>
<td></td>
</tr>
<tr>
<td>Sce₂</td>
<td>×  ×  ×  ×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Sce₃</td>
<td>×  ×  ×  ×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
</tbody>
</table>

ity requirements such as safety or portability. Rather, there are only context-dependent measures, meaningful solely in specific circumstances. Safety for a power plant control software and that for an e-mail client are a fine example. Scenarios offer a way to take context into account since they capture system uses in more specific circumstances [77].

In the running example, the concerns are clearly interlocked. Namely, each scenario expresses multiple FRPs that affect more than one quality attribute. For example, usability is scattered over Sce₁ and Sce₂, and both usability and maintenance are tangled in Sce₁. The same can be said for many functional concerns. Table 5.2 shows how we use scenarios to sort out the relations between FRPs and quality requirements: rows represent scenarios and columns represent FRPs or quality requirements. A check mark, “×”, indicates the requirements construct is included in a particular scenario. Determining such relations is straightforward. Note that the table does not specify the order of FRPs in a scenario; only inclusion relation is considered. Also, as shown in Table 5.1, we use “+” (resp. “−”) to denote a quality requirement is positively (reps. negatively) affected, though finer scales of measuring contributions [106] may be applied.

Although we are primarily interested in the relationship between functional (FRP) and quality (Q) requirements, directly forming a formal context (binary relation) between these two constructs is improbable, as demonstrated by the crosscutting relations in Table 5.2. We exploit scenarios to bridge this gap. In particular, we instantiate formal context by setting the objects (O) to be the FRPs and the attributes (A) to be the
scenarios. The binary relation \((I)\) is given by the inclusion relation, as illustrated in Table 5.2. The basic idea of gluing the third set (quality requirements) to the context is to explore the concept lattice through combinations of overlapping scenarios.

Let’s call the above instantiation \(\text{INS}_1 (O = \text{FRPs}, A = \text{scenarios})\), in which mapping the functional units to objects seems natural. However, three other instantiations of formal context exists:

- \(\text{INS}_2 (O = \text{quality requirements}, A = \text{scenarios})\);
- \(\text{INS}_3 (O = \text{scenarios}, A = \text{FRPs})\); and
- \(\text{INS}_4 (O = \text{scenarios}, A = \text{quality requirements})\).

If we adopt \(\text{INS}_2\), we would employ the same strategy of examining the concept lattice by combining scenarios in order to uncover requirements interactions. \(\text{INS}_3\) and \(\text{INS}_4\) are isomorphic to \(\text{INS}_1\) and \(\text{INS}_2\), respectively, due to the duality principle of lattice theory [14]. In concept analysis, objects and attributes are symmetric so we can switch them without affecting the resulting concept lattice. Objects and attributes are labels used for distinguishing the two sets; the result of concept analysis is determined solely by the binary relation which is independent of these labels [54]. Therefore, in all instantiations, scenarios glue the functional and quality requirements together. No matter which instantiation we choose, the analysis will produce essentially the same result – our view of the concerns is truly symmetric. For the convenience of our discussion in this section, we follow \(\text{INS}_1\).

The concept lattice of our running example is shown in Figure 5.3, where concepts are marked fully by pairs of extents and intents. For extractive SPL requirements analysis, we address two issues concerning aspect-oriented RE: locate quality-specific FRPs and detect interferences. For each issue, we state the problem, present the algorithmic or heuristic solution, and discuss the implications. We will follow a similar presentation style in the next section when addressing the issues in reactive SPL analysis.
– **Issue 1: Locate Quality-Specific FRPs.**

**Problem:** Identify the functional units of the software requirements, i.e., the functional requirements profiles (FRPs), that contribute to a particular system quality.

**Solution:** We classify the concepts in the lattice into different categories according to the quality attribute under investigation. This is achieved by inspecting combinations of overlapping scenarios in the context. The results can be visualized as regions in the lattice.

Let’s take \( Q_1 \) (supporting usability) for example. FRPs specific to \( Q_1 \) can be found in the intersection of the FRPs of the two scenarios \( Sce_1 \) and \( Sce_2 \) because \( Q_1 \) is supported in both these scenarios, as shown in Table 5.2. The intersection of the FRPs for \( Sce_1 \) and \( Sce_2 \) can be identified as the extent of the infimum of the concepts associated with \( Sce_1 \) and \( Sce_2 \): \( (\{FRP_1, FRP_3\}, \{Sce_1, Sce_2\}) \). Since \( Sce_1 \) and \( Sce_2 \) do not share any other quality requirement, the FRPs particularly relevant to \( Q_1 \) are \( FRP_1 \) (navigate shop) and
FRP3 (customize toolbox).

We notice that FRP1 is also used in all other scenarios in the running example, so that one cannot consider FRP1 a specific functional unit for any of Q1, Q2, or Q3. FRP3, in contrast, is used only in scenarios defining Q1. We therefore state the hypothesis that FRP3 is specific to Q1 whereas FRP1 is not. It is worth bearing in mind that this is just a hypothesis because other quality requirements might be involved to which FRP3 is truly specific and that are not explicitly listed in the scenarios. Recall that scenarios are inherently incomplete. Another explanation could be that, by accident, FRP3 is involved in both Q2 (in Sce2) and Q3 (in Sce1); then, it appears in both scenarios but nevertheless is not specific to Q1. However, chances are high that FRP3 is specific to Q1 because FRP3 is not involved when Q2 and Q3 are jointly expressed in Sce3, which suggests that FRP3 at least comes into play only when Q1 interacts with Q2 or Q3. At any rate, the categorization is hypothetic and needs to be validated by the requirements analyst.

FRPs that are somehow related to but not specific for Q1 are such FRPs that are included in scenarios specifying Q1 among other quality requirements. In the running example, both Sce1 and Sce2 define Q1. FRPs in extents of concepts which contain Sce1 or Sce2 are therefore potentially relevant to Q1. In our case, FRP2, FRP4, FRP5, and FRP6 are potentially relevant in addition to FRP1 and FRP3. FRP7 is included only in Sce3, which does not define Q1. Based on the above analysis, we can identify five categories for FRPs according to their relations to Q1. This categorization is displayed in Figure 5.4 by dividing the concept lattice into different regions. We use the sparse representation of the lattice as the classified FRPs are more easily identifiable. The five categories for FRPs with respect to Q1 (supporting usability) are:

- Specific: FRP3 (customize toolbox) is specific to Q1 because it is included in all the scenarios defining Q1 but not in others.

- Relevant: FRP1 (navigate shop) is relevant to Q1 because it is included in all the
scenarios specifying $Q_1$. However, it is more general than FRP$_3$ since it is also included in scenarios not relating to $Q_1$ at all.

- **Pertinent:** FRP$_4$ (select language) and FRP$_6$ (generate report) are included only in scenarios defining $Q_1$. They are less specific than FRP$_3$ because they are not used in all scenarios that define $Q_1$, i.e., these FRPs are only pertinent to $Q_1$. It should be pointed out that, based on the concept lattice, it is not decidable whether the pertinent FRPs (FRP$_4$ and FRP$_6$) are more or less specific than the relevant FRP (FRP$_1$).

- **Shared:** FRP$_2$ (search product) and FRP$_5$ (monitor quantity) are included in scenarios defining $Q_1$ but they are also included in scenarios not defining $Q_1$, i.e., they are shared with other quality requirements. These FRPs are presumably less specific than pertinent and relevant FRPs.

- **Irrelevant:** FRP$_7$ (create account) is irrelevant to $Q_1$ because it is included only in scenarios not specifying the quality requirement $Q_1$. 
The above procedure is amenable to full automation. Our approach not only identifies quality-specific FRPs, but also reveals a relevance ordering regarding the system quality. Four levels of specificity can be inferred: 1) specific, 2) relevant, pertinent, 3) shared, and 4) irrelevant.

Implications: As aforementioned, our treatment of functional and quality requirements in this section is symmetric. Analogously, we can locate quality concerns that are specific to a particular system function. The results can help understand the degree to which requirements concerns correlate and interact. A key problem in aspect-oriented requirement engineering is to identify join points for concerns to be coordinated with. The regions in the concept lattice enable us to focus on more specific concerns, identify join points more accurately, and define pointcuts more efficiently.

Locating concerns within the concept lattice also influences SPL variabilities. When multiple concepts are identified being specific to a concern, we should reconcile and integrate them. If the identified concepts are in a subconcept relation to each other, the superconcept represents a natural merge and is viewed as a strict extension of the behavior of the concern. If, on the other hand, the concepts are incomparable, they may indeed reflect the varying context-dependent behaviors in the SPL, or they may demand more scenarios to be considered so as to discriminate the concerns and disentangle the crosscuts in the lattice.

– Issue 2: Detect Interferences.

Problem: Since the previous issue has addressed interactions between functional and quality requirements, we now concentrate on determining how homogeneous requirements, e.g., a pair of quality attributes, interfere with each other. The discussion is generalizable to functional concerns thanks to our symmetric view of aspects.

Solution: It should be made clear that we regard interference as a syntactic phenomenon. This view is in accordance with Snelting’s work on using concept analysis to restructure
software configurations [150]. Deciding whether or not an interference is harmful is a semantic or pragmatic issue requiring domain knowledge and human judgments. Our goal is to support such decision making by bringing susceptible interferences to light and examining their causes.

In analyzing attributes, coupling arises whenever concepts have objects in common. In [150], two attributes interfere if the intersection of the extent of their concepts in the sparse lattice is not empty. This definition is not directly applicable to our context, but if we leverage the advantage of the concern-locating results described earlier, we are able to adapt the definition as follows: Two quality attributes interfere if the intersection of the specific FRPs contributing to them is not empty.

Let’s use the running example to illustrate the idea. Suppose a novice user expresses her concern that searching products by wild card when navigating the shop is not very effective as mastering the wild card usage is non-trivial. This adds a new scenario, Sce₄,
to our analysis. $\text{Sce}_4$ includes FRP$_1$ and FRP$_2$ (cf. Table 5.1), and specifies $Q_4$: $-U$ (hurting usability), according to the novice user. The updated concept lattice $^2$ is shown in Figure 5.5. If we choose the specificity level up to shared to analyze $Q_1$ and $Q_4$, two interferences can be spotted and shaded in the lattice. Determining trade-offs calls for a deeper, semantic look.

- **FRP$_1$** (navigate shop) is relevant to both $Q_1$ and $Q_4$. In fact, it is the extent of the bottom concept in the lattice, meaning that it is included in all four scenarios in the context. We conjecture that FRP$_1$ is a basic service in the domain. It could be implemented as a utility function and reused in all systems of the product line. Such an interference shows an intentional design so it is not harmful.

- **FRP$_2$** (search product) appears in the intersection of $Q_1$ and $Q_4$ shared regions. This shows coupling between these two concerns and further suggests a join point for coordinating aspect composition. An interesting observation is that $Q_1$ and $Q_4$ are both about usability but they specify opposite ends of the attribute. Semantically, we would expect $Q_1$ and $Q_4$ to be disjoint, so this interference (FRP$_2$) is an inconsistency that is potentially harmful.

- Interference between quality concerns often appears at the terminological level [107], e.g., stakeholders use the same term to mean different concepts. In reviewing the novice user’s scenario, we postulate that her concern may be more accurately characterized as **learnability**. This may induce a subconcept-superconcept relation between learnability and usability, or indicate the interfering FRP behaves variably in the SPL, e.g., “search product by wild card” can hurt learnability, whereas “search product by keyword” may support learnability, depending on the user experience and use context.

$^2$We address the issue of reactively updating the concept lattice in Section 5.3.
Implications: A key benefit of aspect-oriented RE is to detect conflicting concerns early when trade-offs can be resolved more economically [8]. Our work shows that syntactic interferences in the concept lattice can easily be detected automatically. With the domain expert’s additional knowledge, we can make sense of interferences at the semantic and pragmatic levels. Not all interferences are harmful as some show the utility services common in the domain.

Some interferences do have “bad smell”, e.g., those between disjoint or orthogonal concerns [150]. Two concerns are disjoint if they cannot be defined toward the same dimension, e.g., “supporting usability” and “hurting usability”. Two concerns are orthogonal if they deal with independent dimensions of the concern space, e.g., “exception handling” versus “caching and buffering”. Interfering disjoint concerns shows an inconsistency, which indicates a conflict or a varying behavior in the SPL. Interfering orthogonal concerns is also very suspicious: it implies coupling between modules that should be separated rather cleanly.

Detecting interferences helps assess requirements modularity. Based on the analysis of an interference’s cause and effect, the engineer can decide how to react in practice. If it is designed to be there as a service, he may ignore the interference. If it is a term clash or mismatch, he may elicit more scenarios to clarify the subtleties. If it is a variation point, he may capture and model the variabilities in the product line. If it is a conflict, he may prioritize the requirements so he can trade one concern off another. If it is a coupling, he may perform refactoring then use aspect composition to achieve better modularity. At any rate, interferences raise a flag of caution and urges further investigation to crystallize the concepts involved.
5.3 Reactive Requirements Analysis — Asymmetric View of Aspects

When analyzing requirements extracted from existing systems of the product line, we took the symmetric view in Section 5.2 to allow every concern, either functional or quality-related, to express itself in its own dimension. This is a static and micro-level view within the SPL, which focuses on the requirements concerns and their relationships. Although we can form an initial baseline for the SPL, the extraction results are inevitably incomplete so the assets are under-specified. The purpose of reactive SPL development is to enrich the asset base during the SPL’s evolution [83].

In this section, we take the asymmetric view of aspects to analyze the evolving SPL’s requirements. This view distinguishes the base from aspects: The base presents the dominant way of organizing the concerns; the aspects cut across this organization and offer advices to the base at certain join points [8]. An underlying assumption of the reactive model is that there must exist an asset base to react. This base maps naturally to the base in the asymmetric view; the reactive increments, then, are aspects that need to be weaved into the base to enrich the assets. This is a dynamic and macro-level view over the SPL, which focuses on requirements changes and their impacts.

The extractive and reactive models are not isolated pieces but are integral components for flexible SPL development. Thus, the issues discussed in Section 5.2 can help tackle the problems we face in reactive requirements analysis. An example would be aspect weaving, in which a key problem is to identify the join points so that the aspect advises the base at the right places. Issue 1, locating concerns and categorizing them by the specificity ordering, provides a solution: the more specific region in which a point appears, the more intense its interaction with the interested aspect (concern) and the more likely it represents a join point. A pointcut may be formulated by examining patterns of the points within certain relevance regions. Another example is aspect interference, which
concerns about weaving multiple aspects might adversely influence each other’s effect. 
Issue 2, detecting interferences, addresses this problem for obvious reasons.

As was mentioned, the reactive model assumes the existence of the SPL’s requirements assets. Since these assets have undergone in-depth analysis, e.g., the activities described in Section 5.2, they exhibit a high level of quality and stability. It is likely that, at this phase, stakeholders have established a shared domain vocabulary and agreed on the SPL’s architectural drivers \(^3\). The FRPs are also validated and their contributions to the architectural drivers are more directly identifiable. As a result, we instantiate the formal context by setting the objects (\(O\)) to be the FRPs and the attributes (\(A\)) to be the architectural drivers. The binary relation (\(I\)) refers to the contribution relation [106]. Of course, we can (and should) always develop scenarios to help clarify requirements constructs and relations in the context, as discussed in Section 5.2.

The new instantiation (\(O = \text{FRPs}, A = \text{architectural drivers}\)) by no means indicates the relations between functional and quality requirements are fixed. Rather, we anticipate the context to change as the SPL evolves. In particular, we want to react when: 1) new FRPs (e.g., features, services, and system functions) are added to the asset base; and 2) the priority of the architectural drivers changes as the trade-offs are constantly balanced among competing requirements.

\(^3\)A set of quality requirements that guide the SPL’s architecture design (cf. Section 5.1.1). In practice, the number of architectural drivers is often less than a dozen [118].
− **Issue 3:** Update Concept Lattice Incrementally.

**Problem:** Modify the concept lattice efficiently as the SPL evolves. The update should be incremental without having to do a complete re-computation from scratch. As was discussed, we assume the architectural drivers, i.e., the set of attributes \( (A) \), are already identified at this stage, though their priorities may change during the evolution. The major type of modification we consider is in light of new FRPs being added to the context. We plan to investigate other types of modification, such as removing deprecated features, in the near future.

**Solution:** Our solution utilizes Godin et al.’s incremental lattice update algorithm, which takes as input the current lattice \( L \) and the new object with its attributes and outputs a new lattice \( L' \) that incorporates the changes [55]. The efficiency of the algorithm comes from the observation that once an initial concept lattice \( L \) has been constructed from the relation table \( R \), there is no need to maintain \( R \). The incremental lattice update may create new nodes, modify existing nodes, add new edges, and change existing edges that represent the partial ordering \( \leq \). As a result, nodes that were at a specific level \( p \) (where \( p \) is defined as the length of the longest path from the bottom, \( \bot \), to the node) may now be raised in \( L' \) and new nodes may have been created at the level \( p \). However, existing internal nodes will never sink in the lattice because the partial ordering between existing nodes is unchanged by the addition of new nodes [55].

Let \(|O|\) denote the cardinality of the set of objects \( O \), and \(|A|\) denote the cardinality of the set of attributes \( A \). In the worst case, the time complexity for Godin et al.’s incremental algorithm is quadratic in the size of the input relation, i.e., quadratic to \((|O| \times |A|)\) [55]. Our problem, which is concerned only with adding new FRPs (objects), is a special case in which the incremental update is linearly bounded by the number of objects \(|O|\) [55]. Figure 5.6 illustrates the algorithm via the media shop example. The
context is given in Figure 5.6a. The attributes are a set of architectural drivers. For simplicity, we choose the quality attributes appeared in the scenarios (cf. Section 5.1.3), namely, usability (U), accessibility (A), and maintainability (M). Each attribute has two poles (supported and hurt) that are preceded by + and −, respectively.

We use FRP\(_1\) through FRP\(_5\) as baseline requirements, and treat FRP\(_6\) and FRP\(_7\) as incremental updates. The FRP labels were defined in Table 5.1. The concept lattice for baseline FRPs is shown in Figure 5.6b. The bottom (⊥) has intent \{−A\} because, through FRP\(_1\) to FRP\(_5\), no object relates to −A. The top (⊤) has intent \{+U\} because it relates to all the FRPs in the context. The lattice after adding FRP\(_6\) is shown in Figure 5.6c. The newly emerged concept is annotated in stripes. This node is inserted at level 1 as the infimum of ((FRP\(_1\), FRP\(_6\)), \{+M, +A, +U\}) and ((FRP\(_4\), FRP\(_6\)), \{−M, +A, +U\}). The remaining lattice structure of Figure 5.6b is preserved in this modification.

Similarly, Figure 5.6d shows the concept lattice after adding FRP\(_7\). The basic structure of Figure 5.6c remains intact and incremental changes are shaded in crossed stripes. FRP\(_7\) adds the relation to −A in the context, so −A is lifted from the bottom (⊥) to level 1. In the updated lattice, +U is no longer the top (⊤) because FRP\(_7\) has no relation to it. Note that adding new objects does not always result in creating new nodes; new edges may be added or existing nodes and edges may be modified. In all cases, the basic lattice structure is preserved and the increments do not cause the existing nodes to sink in the hierarchy. This provides the linear-order computational efficiency for updating the concept lattice incrementally [55].

Implications: Incorporating incremental changes is essential in reactive SPL development [83]. Adding FRPs to the asset base may be caused by introducing new product functions, upgrading services, eliciting new features, and the like. Updating the concept lattice in a lightweight fashion enables stakeholders to react promptly in order to accommodate the change. We use the term “lightweight” to indicate that our methods can be
used to perform partial analysis on under-specified requirements, without a commitment to developing a complete asset base. Lightweight also means the employed algorithm is so efficient and incremental that a complete lattice re-building can be avoided and the changes can be easily spotted visually.

Our view of reactive analysis is asymmetric: weaving the incremental aspects inevitably affects the structure of the base. This helps assess modularity of the evolving SPL. In Figure 5.6b, for example, +M and −M are kept disjoint, showing a desired low coupling between the poles of maintainability. Adding FRP (generate report) introduces an interference node, as depicted in stripes in Figure 5.6c. Although this seems to destroy the modularity of the base, a semantic case analysis uncovers the SPL’s variability: while generating testing and failure report can ease maintainability, generating consistent sales report may demand extra resources from the maintenance team. In order to perform such assessments and acquire insights to the evolving SPL, quickly and easily identifying the increments is indispensable.


Problem: The general question we address through impact analysis is: Does a change of the fulfillment of a requirement affect the fulfillment of another requirement?

Solution: The solution to the previous issue allows the increments to be recognized easily. This permits automatic generation of sliced lattice according to the change so that a more focused view is obtained. Our approach relies on the subconcept-superconcept relation exposed in the sliced lattice to comprehend the SPL’s requirements change. The solution includes a couple of heuristics:

1). When coping with the change with respect to architectural drivers (quality requirements), adopt a top-down (superconcept to subconcept) strategy. The rationale is that fulfillment of general quality requirement (superconcept) dictates fulfillment of one or more specific requirements (subconcepts).
2). When coping with the change with respect to FRPs (functional requirements), adopt a bottom-up (subconcept to superconcept) strategy. The rationale is that fulfillment of specific function (subconcept) lays the foundation for fulfillment of higher-level requirements (superconcepts).

Figure 5.7 shows a sliced concept lattice, adapted from [106]. Suppose this lattice is the result of some requirements change in the SPL. If the change is about architectural drivers, we would pay more attention to +Accuracy since it appears to be the most general attribute in the hierarchy. This indicates that the change makes SPL’s accuracy a more valuable driver. In the updated context, how to fulfill accuracy drives architectural decisions. For instance, if the SPL’s confidentiality is achieved, so is accuracy. However, if for some reason, e.g., due to conflicting requirements or product variations, confidentiality cannot be fulfilled, we still have an option of implementing the function, search product,
to address the accuracy concern.

If the change is about FRPs, we may start by investigating the concept toward the bottom of the hierarchy. In Figure 5.7, for example, the concept marked with “deploy SSL” is at level 1, which suggests that this FRP is one of the basic services in the SPL. If a product wishes to include higher-level features (superconcepts), such as “enable HTTPS” or “enforce password”, it must select the subconcept, “deploy SSL”, in its configuration. In another word, changing a base-level function’s availability can have a broad impact for members of the SPL.

**Implications:** Comprehending the SPL’s requirements change, as well as the impact of change, plays an important role in balancing trade-offs, determining priorities, and setting preferences. The constraints and dependencies discovered will direct product configurations in the SPL. Our approach takes incremental steps to investigate requirements evolution. The insight comes from exploiting the natural ordering given by the subconcept-superconcept relation in the lattice. However, it is important to keep in mind that change impact analysis requires a deep understanding of the domain. Our heuristics serve as an aid to this understanding and should be treated as such.

### 5.4 Case Study

We used an exploratory case study [166] as the basis for our empirical evaluation. The objective is to assess the usefulness and the scope of applicability of our approach, and more importantly, to identify areas for improvement. In addition, we would like to explore how requirements abstraction and modularity are handled in practice.

#### 5.4.1 Background

The subject system of our study is a commercial mobile soccer game SPL produced by a small software company. In order to honor confidentiality agreements, we will use the
pseudonyms “FC” for the company and “Genuine Soccer” for their SPL. FC started in 1998 with six people specializing in real-time video game development. Genuine Soccer, one of FC’s proprietary systems developed in-house, was first released in 2002 as an embedded game for a specific cell phone provider to exploit the business opportunities offered by the FIFA World Cup™. At first, Genuine Soccer was a single, custom-built product. After serially building different versions and variants of the product, FC began adopting SPL technologies to manage Genuine Soccer in 2005 when the company had approximately 50 employees. The small team size and the short development cycle of Genuine Soccer left little margin for error or resource wastage when performing such a migration.

We studied two variants in the Genuine Soccer family: the Trial version and the Pro version. The screenshot of each version is shown in Figure 5.8. We used the Trial version
to perform extractive analysis. FC shared with us several Trial version’s documents, including the software requirements specification, software design description, and integrated test plan. We applied an information retrieval method [109] to mine FRPs from the documents. A two-hour meeting was held in FC to validate the extracted FRPs, and further elicit the architecturally significant quality requirements from and develop scenarios together with the domain experts. The Pro version was used for reactive analysis. We used the ToscanaJ suite [13] to implement our concept analysis methods. The results were discussed in a half-day joint application development (JAD) workshop in FC.

5.4.2 Results

In preparing the analysis, we identified three stakeholder roles: user, developer, and maintainer. We extracted about 40 FRPs from Genuine Soccer Trial’s project documents. During the first meeting with FC’s experts, we elicited 7 quality requirements as the mobile game SPL’s architectural drivers. It is interesting to note that these drivers include not only typical software engineering quality attributes like performance and modifiability, but also emotional requirements in video games like excitement and frustration [24]. We collaborated with the domain experts and devised 12 scenarios, making use of 17 FRPs to define the architectural drivers. For each role, 4 scenarios were

<table>
<thead>
<tr>
<th>Constructs</th>
<th>#</th>
<th>Concerns</th>
<th>#</th>
<th>Pre.*</th>
<th>Interferences</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>Roles</td>
<td>3</td>
<td>Specific</td>
<td>2.7</td>
<td>88.9%</td>
<td>Utility</td>
<td>2</td>
</tr>
<tr>
<td>FRPs</td>
<td>17</td>
<td>Rel., Pert.*</td>
<td>6</td>
<td>78.9%</td>
<td>Conflict</td>
<td>2</td>
</tr>
<tr>
<td>Drivers</td>
<td>7</td>
<td>Shared</td>
<td>6</td>
<td>57.5%</td>
<td>Variability</td>
<td>0</td>
</tr>
<tr>
<td>Scenarios</td>
<td>12</td>
<td>Irrelevant</td>
<td>2.3</td>
<td>54.4%</td>
<td>Coupling</td>
<td>5</td>
</tr>
</tbody>
</table>

* Precision

* Relevant, Pertinent
elicited. Each scenario helped specify 1 to 3 quality attributes, and was expressed within the 5-sentence narratives. A brief summary of this setup is shown in the left column of Table 5.3.

To evaluate the effectiveness of the concern-locating results (Issue 1, cf. Section 5.2), we used 3 architectural drivers (positive poles): excitement, performance, and modifiability. These requirements were chosen in order to have a fair coverage of various stakeholders' interests. For each driver, we ranked the 17 FRPs in the context by the specificity ordering introduced in Section 5.2: 1) Specific, 2) Relevant, Pertinent, 3) Shared, and 4) Irrelevant. The number of concerns categorized by specificity levels (average of three architectural drivers) is shown in the middle of Table 5.3. Meanwhile, we asked the experts to manually identify the quality-specific units among the 17 FRPs. This allows us to compute precision [140] of our results. Precision measures accuracy and is defined as the proportion of located concerns which are relevant. The average precision achieved at each specificity level is listed in Table 5.3.

The findings demonstrate the usefulness of our concern-locating method and the specificity ordering, since higher precision is achieved at more specific levels. Ideally, we would like to also compute recall [140], the proportion of located relevant concerns to the total number of all relevant concerns, in order to measure the coverage of our results. In this case study, recall values were not obtained. On one hand, the incomplete nature of scenarios would not favor the coverage measure. For example, among the 40 or so FRPs extracted from the Genuine Soccer Trail project documents, only 17 appeared in the context used for the analysis. On the other hand, the under-specifying principle [83] challenges the value of measuring coverage for extractive SPL engineering. While further testing is in order, current results showed that our method could accurately identify the relevant concerns.

Based on the concern-locating results, we chose the specific-level FRPs of each architectural driver for detecting interferences (Issue 2, cf. Section 5.2). We presented
the concept lattice to FC’s experts, and discussed 9 interferences and their causes and effects (right column of Table 5.3). We were unable to find the SPL’s variability; part of the reason was due to one product (Genuine Soccer Pro) being a strictly enhanced version of the other (Genuine Soccer Trial). We expect the results to be supplementary, i.e., more variations would be identified, when reconciliation, instead of consolidation, is prevalent in merging artifacts of the SPL. Among the 9 interferences, we identified more than half as coupling. We ended up recommending a restructure or refactoring of the requirements portion so as to improve modularity. The results also included a few utilities and terminological conflicts. Through interviews, the experts confirmed the effectiveness of visualizing the overlaps in the concept lattice.

Genuine Soccer Pro shares most system functions with its Trial counterpart, such as setting line up and displaying score. Notably, it has a bunch of advanced features, e.g., showing weather conditions and controlling detailed player moves like bicycle kick, elastico, and penetrative pass. We used some of these features for reactive analysis: updating lattice and analyzing change impact (cf. Section 5.3). The results were presented in the JAD workshop, and the feedback was very positive regarding the usefulness of the heuristics and the capability of accommodating requirements changes.

Several factors can affect the validity of the exploratory case study [166]. As for construct validity, for example, the interpretation of concern “specificity” may vary among experts, but we believe visualizing lattice regions could overcome this limitation. As for external validity, Genuine Soccer Pro is an enhancement of the Trial version, so our reactive analysis focused on adding features. Other reactive modifications may exhibit different properties, but should not change the key insights from our study significantly. As for reliability, we expect that replications of the study would offer similar results. Of course, the characteristics of each SPL under study will differ from our reports.
5.4.3 Discussion and Lessons Learned

We are confident that the proposed framework is scalable because: 1) the algorithmic solutions are in polynomial time, 2) the implementation has been leveraged from off-the-shelf toolkits (e.g., ToscanaJ [13]), and 3) the mobile-game case is reasonably sized among SMEs (small and medium-sized enterprises), which are the main beneficiaries of the extractive SPL model [83]. In practice, more focused extraction can be carried out first, followed by reactive increments.

However, manual effort is indispensable in our framework, which includes devising quality attribute scenarios, identifying architectural drivers, teasing out the relationship among requirements artifacts and scenarios, reasoning about interferences, etc. Although we are keen to research more heuristics, guidelines, and automation support in these respects, domain expertise is essential since our framework is not a replacement, but a complement, to existing SPL methods.

We now share some key observations from meeting and interviewing FC’s experts. We found scenario generation is much like software testing: We cannot prove we have a sufficient number of test cases, but we can determine a point at which adding new test cases yields only negligible improvement. In practice, the available resource is another factor. We devised 4 scenarios for each of the 3 stakeholder roles in half an hour, which seemed adequate for our analysis. Having a set of FRPs extracted from requirements documents [108] enabled us to generate scenarios more easily.

We realized that our methods should be applied iteratively, not in a strict “extractive then reactive” manner. For example, even though no variability was identified in examining Genuine Soccer Trial’s requirements, when adding new features to the SPL, variations emerged and so did other types of interference. This requires efficient updates to catch and visualize the change and impact, which our lightweight approach offers.

“Crosscut follows form; form follows function” was FC’s view of modularity in practice. The Genuine Soccer SPL’s requirements were written in a textual form following...
IEEE-STD-830 standard [17], which had an emphasis on system functionalities. Quality requirements inevitably cut across this form. Teasing out basic constructs like FRPs and analyzing requirements interaction on the fly were not just an academic exercise, but something FC considered incorporating in daily practice because they could identify many potential problems early in the software life cycle and at an extremely low cost.

On an organizational level, coupling is not necessarily a bad thing – it decreases latency, according to FC. The company enjoyed the culture of small project teams, supported by their flat organizational structure. They learned the lesson that the organizational structure can artificially reduce the throughput of business processes. A structure that caused information to flow through many roles not only increased latency, but also caused loss of information fidelity. FC’s experience suggested that taking advantage of coupling was to open communication paths between roles to increase the overall coupling/role ratio, particularly between central process roles.

### 5.5 Related Work

The work presented in this chapter is related to several different efforts. Baniassad et al. were among the first to discover early aspects and exploit them throughout the software development life cycle [8]. The benefit is to detect conflicting concerns early, when trade-offs can be resolved more economically. Their view is asymmetric in that aspects depend on the author’s chosen organization of requirements artifacts. A taxonomy of asymmetric requirements aspects is described in [111].

The symmetric view of aspects was formulated in [153], but most contemporary SPL methods still exhibit “tyranny of the dominant decomposition” [153]: they use either functional or quality requirements as the primary criteria for organizing and structuring. We argued that both asymmetric and symmetric views can be leveraged to understand the SPL’s requirements modularity and interactions. In particular, we described how
isomorphic context instantiations could result in a symmetric view of concerns.

Likewise, our work is related to the effort of managing SPL feature relation and dependency via aspects [29]. The authors introduced some aspect-oriented patterns for implementing variable features on top of an object-oriented (OO) approach. Similarly, Liu et al. [91] used aspects to address the crosscutting variabilities in an OO SPL design. Both work demonstrated the benefit that, by exploiting aspects, inclusion or exclusion of variable SPL features would cause little change to components implementing or modeling other features. To support aspect tracing from requirements to implementation and testing, we contributed a goal-driven framework that made aspects discovered in requirements analysis become true engineering assets [112]. In [158], the authors used a traceability matrix to identify aspects. In [173], the authors validated the aspects discovered in KAOS requirements model [42] through code aspects.

Most research into SPL requirements assets has worked in a proactive fashion. Recently, effort has been made in order to extract requirements assets from software repositories. Natural language processing and information retrieval techniques have attracted much attention (e.g., [141, 2]), since legacy requirements are mostly documented in natural language. Semantic analysis was also performed to model SPL variabilities [108] and to intentionally compose the aspectual requirements [28]. In terms of reactive development, Baldwin and Coady discussed their experience of introducing a distribution implementation as aspects into the JVM [7]. The aspect-oriented increments improved the internal code structure and made external interactions explicit. Our work deals with the increments in order to enrich the requirements assets in reactive SPL engineering.

Concept analysis [54] has traditionally been applied in the field of software engineering to support software maintenance activities [154], such as program understanding [155], reengineering configuration structures [150], and code-level aspect mining [156]. In the analysis of software systems, especially source code exposing certain structural and behavioral properties, several relationships among the composing entities emerge. For this
reason, concept analysis has found a very productive application area associated with software reengineering and program comprehension. Recent work, such as [132], has also reported the application of concept analysis in requirements engineering activities. Our earlier work exploited structural properties in requirements goal models to analyze aspects [106]. The current endeavor refines and extends our proposal, thereby enhancing the overall competence of concept analysis in investigating crosscutting properties in early phases of software development.

5.6 Summary

In this chapter, we contributed a lightweight conceptual framework to analyze the requirements assets in support of extractive and reactive SPL development. By taking advantage of both symmetric and asymmetric views of aspects, we were able to ameliorate “tyranny of the dominant decomposition” [153]. We used scenarios in defining the context to bridge functional and quality requirements. Their relationships are organized in a concept lattice, which provides a richer notion of modularity for understanding requirements abstractions. We formulated a number of issues faced in aspect-oriented product line requirements engineering, presented our solutions according to the concept lattice, and discussed the implications. Studying a mobile game SPL demonstrated the applicability and usefulness of our approach. Our future work involves several different strands. First, more in-depth empirical studies are needed to lend strength to the preliminary findings reported here. Second, more user-friendly interpretations of the lattice, which shield the user from the mathematical complexities, are in order. Third, we would like to support other types of reactive changes besides adding new product functions.
Chapter 6

Applications of FRPs in Product Line Engineering

In this chapter, we present two applications: on-demand cluster analysis [109] and detecting terminological interferences [107], to show how FRPs can be used to support other activities in SPL engineering. Both applications take advantage of the FRPs’ primitive level of abstraction. FRPs capture the domain’s action themes, so they represent the concerns at different abstraction levels from other RE constructs like goals, intentions, and constraints. In linguistics, It is argued that any semantically complex word can be explicated by means of an exact paraphrase composed of simpler, more intelligible words (known as semantic primitives) than the original [162]. In this sense, we believe FRPs are requirements primitives from the functionality’s perspective. They help us understand the actions that need to be performed by the software-to-be.

6.1 On-Demand Cluster Analysis

Many researchers have attempted to organize and find structure in requirements, giving rise to the research area of requirements clustering. A clustering algorithm tries to group individuals into classes/clusters, i.e., a set of requirements that are in some way
characterized by an internal coherence and/or an external isolation. Clustering is an example of unsupervised learning, meaning it does not rely on predefined classes and manually-labeled training set [60]. For this reason, clustering is readily amenable to automation, and thus fits our framework’s lightweight philosophy.

Before clustering, we must decide the objects to be clustered and the attributes used to determine the relationship between the objects. Current requirements clustering approaches seldom consider the homogeneity and the granularity level of objects, causing insensible clustering results. They also rarely seek automatic support for uncovering semantic attributes. To overcome these shortcomings, we have developed a semi-automated technique for extracting the SPL’s functional requirements profiles (FRPs) from natural language documents [108]. FRPs, which capture the domain’s action themes at a primitive level, are our clustering objects. One contribution of our clustering approach is to leverage Fillmore’s case theory [50] to systematically identify the semantic attributes associated with each FRP.

After clustering, the results are subject to analysis. Hartigan [61] described some general activities that the cluster analyst should perform: name the clusters, use them to help summarize and display the data, and explain in evident groupings. We realize that, when analyzing SPL requirements, stakeholders have different goals. One clustering algorithm can hardly satisfy all the needs. We therefore propose on-demand analysis to take different perspectives into account. Hartigan’s tasks are refined in this section for specific stakeholder roles. From the user’s viewpoint, clustering helps identify, browse, and prioritize features. From the designer’s viewpoint, clustering helps achieve system decomposition and modularization.

This section introduces, to the best of our knowledge, the first on-demand cluster analysis framework for disentangling stakeholder goals. Our work tackles some key aspects of RE in SPLs, including feature identification, information encapsulation, requirements prioritization, and variability management. We also consolidate and advance the
requirements clustering literature by exploiting overlapping clusters and by investigating clusters that minimize the information loss. We study an order-processing SPL to demonstrate the applicability and usefulness of our approach. The study shows that clusters, by defining higher-level requirements according to stakeholder intentions, can bring SPL engineering the benefits of dealing with abstractions: they provide a context, reduce complexity, facilitate communication, and enhance modularization.

6.1.1 Motivating Example

We adapt the library management information system (MIS) [26, 165] as our running example. Figure 6.1a shows a partial requirements list for the library MIS. This list is clearly too trivial to require clustering. However, it suffices to illustrate our approach’s clustering objects, i.e., functional requirements profiles (FRPs), and their semantic attributes. A fuller study is presented in Section 6.1.4.

Figure 6.1b lists the example’s FRPs. Not all verb-DO pairs are FRPs. In Chapter 3, we define an entropy-based measure to facilitate FRPs selection. The quantified information value helps determine requirements at a primitive level. According to Moon et al. [100], a primitive requirement (PR) is a transaction that has an effect on an external
actor; sample PRs for the online news domain are “write an opinion” and “forward an article by e-mail”. We enhance the idea of PR by heeding linguistic constructs in the domain of discourse. Our goal is to identify a set of concrete functional units at the same granularity level that yield some observable results of value to an external actor. In practice, if the verb-DO pair contains a too general element (e.g., “handle”, “manage”, “information”, etc.), then it is unlikely to be a proper FRP. This is similar to many linguistic RE methods’ (e.g., [116, 70]) efforts of filtering out the stop words [47] from the extracted requirements constructs.

Domain knowledge plays a key role in producing FRPs. For example, R5 in Figure 6.1a not only implies that the DO of “search” is a “checkout” list, but also unfolds another FRP: “check out book”. Knowledge about the library MIS also helps refine the lexical pair “add user” in R7 into FRP8 and FRP9 in Figure 6.1b. We use Fillmore’s case theory [50] as a basis for understanding language semantics for the FRPs, as described in Chapter 4. Three essential attributes (cases) for the FRP are considered:

- **Agentive** defines the agent(s) whose activities will bring about the FRP’s state of affairs. Responses to this question are typically actors or combinations of actors found in the domain, including the system-to-be, e.g., \{borrower, librarian\}_{Agentive} “search checkout”.

- **Objective** defines the object(s) that is affected by the FRP’s activity. Since DO is already part of the FRP, this case concerns mainly with the types of DO involved, e.g., “search \{overdue, admissible\}_{Objective} checkout”.

- **Conditional** defines the trigger(s) of the FRP’s action or the condition(s) under which the FRP’s function can be achieved, e.g., “search checkout” only if \{“identify borrower”, “check authentication”\}_{Conditional}.

The rationale of choosing the above cases is twofold. First, they represent *contextual* variation dimensions, as opposed to *intentional* ones [90]. Contextual cases, i.e., variables
that exist in the problem situation, are intrinsic factors in RE because they shape and constrain the intentional cases. Second, they hinge on the candidate conceptual classes and attributes for the purpose of domain analysis and software design [128]. Manipulating the agents, objects, and conditions crystallizes the facets within the FRP’s semantic frame and, more importantly, the above cases serve as the reference points for unraveling the relationships between the FRPs.

The set of essential attributes is by no means a template for structuring functional requirements descriptions, but a catalogue of categories that can help analysts understand the key variability aspects of functional requirements. Figure 6.2 illustrates the semantic attributes of two FRPs, in which domain entities and interdependencies are uncovered. Concentrating on a small number of attributes speeds up domain analysis. Meanwhile, our work on modeling FRP’s variabilities [108], along with reported linguistic RE tools (e.g., [116, 80]), can automate, to a great extent, this uncovering process. The results from any automated toolset must, of course, be validated by the domain experts.

We organize the FRPs and all relevant attributes in a matrix, as shown in Table 6.1. To simplify the example, we use general attributes $A_1$ to $A_9$ in Table 6.1. One may fine-tune the attributes with respect to the cases. For example, “$A_2$: borrower” can be replaced by “borrower_{Agentive}” and “borrower_{Objective}”. FRP$_1$ to FRP$_9$ appeared earlier in Figure 6.1b. When reviewing semantic frames, experts have decided to add the assets “FRP$_{10}$: check authentication” and “$A_9$: ID & password”.

![Table 6.1: Semantic attributes (library MIS)](image-url)
Table 6.1: FRP-attribute matrix (cf. Figure 6.1b)

<table>
<thead>
<tr>
<th>A₁ : system-to-be</th>
<th>FRP₁</th>
<th>FRP₂</th>
<th>FRP₃</th>
<th>FRP₄</th>
<th>FRP₅</th>
<th>FRP₆</th>
<th>FRP₇</th>
<th>FRP₈</th>
<th>FRP₉</th>
<th>FRP*₁₀</th>
</tr>
</thead>
<tbody>
<tr>
<td>A₂ : borrower</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>A₃ : librarian</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>A₄ : administrator</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>A₅ : book</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A₆ : reservation</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A₇ : checkout</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A₈ : e-mail address</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A₉ : ID &amp; password</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

* FRP*₁₀ : check authentication

In the matrix, we indicate the presence of attributes with 1 and their absence with 0. Note that, for a given FRPᵢ, Aᵢ is present if Aᵢ is an essential attribute for FRPⱼ, i.e., FRPⱼ’s action critically depends on Aᵢ. Again, one may polish the decision on how to determine the presence or absence of a feature without affecting the rest of our discussion.

### 6.1.2 User-Centered Clustering

#### Stakeholder Goals

From the user’s point of view, clustering shall help determine external utility and system acceptance, as opposed to decide an internal implementation structure [165]. The external view is essential to a SPL’s success [58], since it allows a system to be partitioned into usable subsystems for the user to fulfill part of her missions. We consider three activities: feature identification, browsing, and prioritization, to help achieve stakeholder goals in user-centered clustering.

A feature describes a product characteristic from user or customer views [75], which essentially consists of a cohesive set of individual requirements [157]. In our approach, a feature represents a functional abstraction [76] and is identified by grouping tightly related FRPs. Browsing allows users to better understand and retrieve relevant features.
Table 6.2: Dissimilarity indices (δ) of Table 6.1

<table>
<thead>
<tr>
<th></th>
<th>FRP₁</th>
<th>FRP₂</th>
<th>FRP₃</th>
<th>FRP₄</th>
<th>FRP₅</th>
<th>FRP₆</th>
<th>FRP₇</th>
<th>FRP₈</th>
<th>FRP₉</th>
<th>FRP₁₀</th>
</tr>
</thead>
<tbody>
<tr>
<td>FRP₁</td>
<td>0.00</td>
<td>0.25</td>
<td>0.60</td>
<td>0.80</td>
<td>0.67</td>
<td>0.60</td>
<td>0.40</td>
<td>0.67</td>
<td>0.86</td>
<td>0.67</td>
</tr>
<tr>
<td>FRP₂</td>
<td>0.00</td>
<td>0.80</td>
<td>1.00</td>
<td>0.60</td>
<td>0.80</td>
<td>0.60</td>
<td>0.83</td>
<td>0.60</td>
<td>0.60</td>
<td>0.60</td>
</tr>
<tr>
<td>FRP₃</td>
<td>0.00</td>
<td>0.83</td>
<td>0.80</td>
<td>0.83</td>
<td>0.60</td>
<td>0.83</td>
<td>0.83</td>
<td>0.83</td>
<td>0.83</td>
<td>0.83</td>
</tr>
<tr>
<td>FRP₄</td>
<td>0.00</td>
<td>0.60</td>
<td>0.40</td>
<td>0.67</td>
<td>0.67</td>
<td>0.67</td>
<td>0.67</td>
<td>0.67</td>
<td>0.67</td>
<td>0.67</td>
</tr>
<tr>
<td>FRP₅</td>
<td>0.00</td>
<td>0.83</td>
<td>0.83</td>
<td>1.00</td>
<td>0.83</td>
<td>0.83</td>
<td>0.83</td>
<td>0.83</td>
<td>0.83</td>
<td>0.83</td>
</tr>
<tr>
<td>FRP₆</td>
<td>0.00</td>
<td>0.67</td>
<td>0.67</td>
<td>0.83</td>
<td>0.83</td>
<td>0.83</td>
<td>0.83</td>
<td>0.83</td>
<td>0.83</td>
<td>0.83</td>
</tr>
<tr>
<td>FRP₇</td>
<td>0.00</td>
<td>0.40</td>
<td>0.40</td>
<td>0.40</td>
<td>0.40</td>
<td>0.40</td>
<td>0.40</td>
<td>0.40</td>
<td>0.40</td>
<td>0.40</td>
</tr>
<tr>
<td>FRP₈</td>
<td>0.00</td>
<td>0.40</td>
<td>0.40</td>
<td>0.40</td>
<td>0.40</td>
<td>0.40</td>
<td>0.40</td>
<td>0.40</td>
<td>0.40</td>
<td>0.40</td>
</tr>
<tr>
<td>FRP₉</td>
<td>0.00</td>
<td>0.40</td>
<td>0.40</td>
<td>0.40</td>
<td>0.40</td>
<td>0.40</td>
<td>0.40</td>
<td>0.40</td>
<td>0.40</td>
<td>0.40</td>
</tr>
<tr>
<td>FRP₁₀</td>
<td>0.00</td>
<td>0.40</td>
<td>0.40</td>
<td>0.40</td>
<td>0.40</td>
<td>0.40</td>
<td>0.40</td>
<td>0.40</td>
<td>0.40</td>
<td>0.40</td>
</tr>
</tbody>
</table>

Prioritizing clusters facilitates the discovery of mandatory and optional features, thereby establishing an incremental delivery strategy for the SPL.

**Overlapping Clustering**

Most requirements clustering methods in the literature identify a requirement as a member of one and only one cluster. This classification of requirements into mutually exclusive and collectively exhaustive clusters, although methodologically elegant, is conceptually questionable. Requirements, especially aspectual (crosscutting) ones [111], can belong to more than one cluster. Furthermore, if we think of a cluster as a feature, then the feature interaction problem [68] shows that some clusters are not disjoint.

We argue that an overlapping cluster analysis is desirable to map requirements onto the external end-user’s view. The OPC (overlapping partitioning cluster) algorithm [27] is adapted to partition the FRPs into clusters that may overlap with each other. We first use the FRP-attribute matrix, illustrated in Table 6.1, to automatically derive dissimilarity indices, numerical measures used as the basis to determine clusters [61].

Let Ω be a set of objects. A dissimilarity index δ over Ω² is a function from Ω × Ω to \( R_+ \) that satisfies two properties: 1) \( \forall o \in \Omega, \delta(o,o) = 0 \), 2) \( \forall (o,o') \in \Omega², \delta(o,o') = \delta(o',o) \).
Table 6.3: Similarity levels (σ) of Table 6.2

<table>
<thead>
<tr>
<th></th>
<th>FRP_1</th>
<th>FRP_2</th>
<th>FRP_3</th>
<th>FRP_4</th>
<th>FRP_5</th>
<th>FRP_6</th>
<th>FRP_7</th>
<th>FRP_8</th>
<th>FRP_9</th>
<th>FRP_10</th>
</tr>
</thead>
<tbody>
<tr>
<td>FRP_1</td>
<td>1.00</td>
<td>0.74</td>
<td>0.38</td>
<td>0.17</td>
<td>0.30</td>
<td>0.38</td>
<td>0.58</td>
<td>0.30</td>
<td>0.10</td>
<td>0.30</td>
</tr>
<tr>
<td>FRP_2</td>
<td></td>
<td>1.00</td>
<td>0.17</td>
<td>0.38</td>
<td>0.17</td>
<td>0.74</td>
<td>0.38</td>
<td>0.14</td>
<td>0.38</td>
<td></td>
</tr>
<tr>
<td>FRP_3</td>
<td></td>
<td></td>
<td>1.00</td>
<td>0.00</td>
<td>0.38</td>
<td>0.17</td>
<td>0.38</td>
<td>0.14</td>
<td>0.38</td>
<td></td>
</tr>
<tr>
<td>FRP_4</td>
<td></td>
<td></td>
<td></td>
<td>1.00</td>
<td>0.14</td>
<td>0.14</td>
<td>0.38</td>
<td>0.14</td>
<td>0.14</td>
<td></td>
</tr>
<tr>
<td>FRP_5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.00</td>
<td>0.38</td>
<td>0.58</td>
<td>0.30</td>
<td>0.30</td>
<td>0.58</td>
</tr>
<tr>
<td>FRP_6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.00</td>
<td>0.14</td>
<td>0.14</td>
<td>0.00</td>
<td>0.14</td>
</tr>
<tr>
<td>FRP_7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.00</td>
<td>0.30</td>
<td>0.30</td>
<td>0.14</td>
</tr>
<tr>
<td>FRP_8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.00</td>
<td>0.58</td>
<td>0.58</td>
</tr>
<tr>
<td>FRP_9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.00</td>
<td>0.58</td>
</tr>
<tr>
<td>FRP_10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.00</td>
</tr>
</tbody>
</table>

Since our data of the FRP-attribute matrix is on a nominal scale \(^1\), we use a simple matching mechanism \(^60\) to compute the dissimilarity index. Specifically, for any pair of FRP\(_i\) and FRP\(_j\), \(\delta_{ij} = \delta(\text{FRP}_i, \text{FRP}_j) = (p - m)/p\), where \(p\) is the total distinct attributes in the two FRPs and \(m\) is the number of matching attributes between the FRPs. \(^2\) Table 6.2 shows the dissimilarity indices of the running example, in which \(\delta_{ij} \in [0, 1]\) for all FRP\(_i\) and FRP\(_j\).

Next, the similarity level \(^27\) is defined as: \(\sigma_{ij} = 1 - \min\{\delta_{ij}, \delta_{if}\}/\delta_{if}\), where \(\delta_{if}\) is the top 5% percentile of dissimilarity indices of all \(\delta_{ij}\) pairs. In this definition, if \(\delta_{ij} \geq \delta_{if}\), the similarity between FRP\(_i\) and FRP\(_j\) will be 0. The advantage of this definition is to make the similarity more robust to outliers or extreme values \(^27\). Table 6.3 shows the running example’s similarity levels with \(\delta_{if} = 0.96\).

The OPC algorithm \(^27\) requires the user to specify two parameters: the number of desired clusters \(k\), and the minimum threshold value for similarity level \(\sigma\). Through iterative adjustments, OPC attempts to balance two related, and potentially conflicting,

---

\(^1\)A nominal variable is a generalization of the binary variable in that it can take on more than two states \(^60\). The states can be denoted by numbers, but the numbers do not represent any specific ordering.

\(^2\)According to \(^60\), weights can be assigned to increase the effect of \(m\) or the FRPs associated with a larger number of attributes; however, the current work does not make use of any weighting schemes.
objectives: maximizing the average number of objects in a cluster, and maximizing the
distances among cluster center objects. We change OPC by not requiring a fixed \( k \) since
it is hard to accurately define the number of clusters \emph{a priori}. Instead, we allow the
user to adjust \( \sigma \) values to explore features with different granularities. Meantime, OPC’s
objective of maximizing the distances among cluster centers is maintained. The changed
algorithm can be run in time of \( O(n^2) \), where \( n \) is the number of objects (FRPs) \[27\].

In the library MIS, domain knowledge helps the user choose \( \sigma=0.50 \) to analyze FRP-
clusters. Taking each FRP \( i \) as the central object of the cluster \( C_\alpha \), FRP \( j \) belongs to \( C_\alpha \) if
\( \sigma_{ij} \geq \sigma \). In Table 6.3, the \( \sigma_{ij} \) values that satisfy the above condition are shown in boldface.
To achieve OPC’s objectives, cluster’s center and size are iteratively adjusted \[27\]. The
final result of the running example consists of six clusters, among which three have more
than one FRP \(^3\):

\[
\begin{align*}
C_1 &= \{\text{FRP}_1, \text{FRP}_2, \text{FRP}_7\} \\
C_2 &= \{\text{FRP}_5, \text{FRP}_7, \text{FRP}_{10}\} \\
C_3 &= \{\text{FRP}_8, \text{FRP}_9, \text{FRP}_{10}\}
\end{align*}
\]

Based on the clustering result, both Hartigan’s tasks \[61\] and the activities defined
earlier can be carried out. The three clusters shown above represent the features iden-
tified at a higher abstraction level than the individual FRPs. Figure 6.3 summarizes
and displays the overlapping clusters. Such a non-hierarchical representation facilitates
browsing the features and their functional constituents. In this example, we are also able
to name the clusters: \( C_1 \) handles “reservation” services, \( C_2 \) defines “searching” facilities,
and \( C_3 \) gathers “admin” duties. Figure 6.3 annotates these feature names and places
single-FRP clusters (\( \{\text{FRP}_3\} \), \( \{\text{FRP}_4\} \), and \( \{\text{FRP}_6\} \)) near the FRPs having a relatively
high similarity level. One reason that these FRPs are isolated is due to the incomplete
requirements list (cf. Figure 6.1a). However, if the user changes the threshold value of

\(^3\)The final result is different from simply inspecting Table 6.3 and taking each FRP \( i \) as center in turn. For example, the cluster centered around FRP \( 7 \) is rejected because it does not produce the maximal distance among the centers. The detailed OPC adjustments can be found in \[27\].
σ, new clusters may be formed or more FRPs may become isolated.

A major advantage of overlapping clustering is to make explicit the crosscutting concerns. In Figure 6.3, FRP₇ can be localized in C₂ as a searching facility and then be invoked by the reservation feature (C₁). FRP₁₀, on the other hand, is a requirements aspect [111] since this security concern tends to cut across multiple features in the system. Factoring out FRP₁₀, along with other security requirements, in a separate module can ameliorate concerns’ tangling and scattering, thereby improving requirements’ comprehensibility.

Finally, user-centered feature prioritization can help discover variabilities. As pointed out by Pohl et al. [125], features that have a high priority for some stakeholders but a low priority for others are candidates for variable requirements in the SPL. Another method is to ask the user to prioritize how attractive a feature is [126]. Basic features define core functionalities (e.g., FRP₆ in the library MIS) whose absence leads to customer dissatisfaction. These dissatisfiers are thus mandatory features in the SPL. In contrast, satisfiers or delighters stand for nice-to-have or visionary functionalities (e.g., FRP₇ in the library MIS), so they are optional features in the SPL [126].
6.1.3 Design-Driven Clustering

Stakeholder Goals

From the designer’s point of view, clustering shall help create a meaningful decomposition of the system into smaller, more manageable subsystems/modules. One of the most widely accepted quality objectives for design is modularity [119]. In our approach, the degree of a design’s modularization is implied by both intra-cluster cohesion and inter-cluster coupling.

The overlapping clustering described in Section 6.1.2 can assist the designer in conceptualizing the system requirements, and more importantly, teasing out the aspectual requirements that are unlikely to be separated cleanly in traditional decompositions. The designer has to pay special attention to these crosscutting concerns. The remaining requirements, for practical reasons, shall not overlap in the system design. Decomposing requirements necessarily drives design decisions to be made.

A benefit of clustering is reducing software complexity by replacing a set of artifacts with a cluster, a representative abstraction of all artifacts grouped within it. Thus, the obtained decomposition is easier to understand. The flipside of the coin, however, is that the amount of information conveyed by the clustered representation is also reduced. Our main objective is to minimize information loss during the design-driven clustering process.

Information-Theoretic Clustering

We adopt an information-theoretic clustering algorithm [3] to build design modules. The original idea in [3] focused on reverse engineering the architecture of legacy systems: program files were clustered based on structural dependencies (e.g., call graphs) and nonstructural properties (e.g., file names). Our task is forward engineering a SPL’s design baseline from the FRPs. The information carried by each FRP is indicated by its
Table 6.4: Normalized matrix of Table 6.1

<table>
<thead>
<tr>
<th></th>
<th>FRP_1</th>
<th>FRP_2</th>
<th>FRP_3</th>
<th>FRP_4</th>
<th>FRP_5</th>
<th>FRP_6</th>
<th>FRP_7</th>
<th>FRP_8</th>
<th>FRP_9</th>
<th>FRP_10</th>
</tr>
</thead>
<tbody>
<tr>
<td>A_1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1/3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A_2</td>
<td>1/4</td>
<td>1/3</td>
<td>0</td>
<td>1/3</td>
<td>1/4</td>
<td>1/4</td>
<td>0</td>
<td>1/4</td>
<td>0</td>
<td>1/4</td>
</tr>
<tr>
<td>A_3</td>
<td>0</td>
<td>0</td>
<td>1/3</td>
<td>0</td>
<td>1/4</td>
<td>0</td>
<td>1/4</td>
<td>0</td>
<td>1/4</td>
<td>0</td>
</tr>
<tr>
<td>A_4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1/4</td>
<td>1/4</td>
<td>0</td>
</tr>
<tr>
<td>A_5</td>
<td>1/4</td>
<td>0</td>
<td>1/3</td>
<td>0</td>
<td>0</td>
<td>1/3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A_6</td>
<td>1/4</td>
<td>1/3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1/4</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A_7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1/4</td>
<td>1/3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A_8</td>
<td>0</td>
<td>0</td>
<td>1/3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1/4</td>
<td>1/4</td>
<td>0</td>
<td>1/4</td>
</tr>
<tr>
<td>A_9</td>
<td>1/4</td>
<td>1/3</td>
<td>1/3</td>
<td>0</td>
<td>1/4</td>
<td>0</td>
<td>1/4</td>
<td>1/4</td>
<td>1/4</td>
<td>1/4</td>
</tr>
</tbody>
</table>

association with the semantic attributes identified in the domain.

We now introduce some basic concepts from information theory [38], and use the running example to illustrate the clustering algorithm in [3]. Let $A$ denote a discrete random variable taking its values from a set of objects $\mathcal{A}$. In our example, $\mathcal{A}$ is the set \{FRP_1, FRP_2, \ldots, FRP_{10}\}. \footnote{For illustrative purposes, the aspectual requirement FRP_{10} is considered together with the other FRPs.} If $p(a_i)$ is the probability mass function of the values $a_i$ that $A$ takes ($a_i \in \mathcal{A}$), the entropy of variable $A$ is defined by: $H(A) = -\sum a_i p(a_i) \log_2 p(a_i)$. Intuitively, entropy is a measure of disorder: the higher the entropy, the lower the certainty with which we can predict the value of $A$ [38].

Let $B$ be a second random variable over the set $\mathcal{B}$ of all the attributes. $\mathcal{B}$ is the set $\{A_1, A_2, \ldots, A_9\}$ in our example, as given in Table 6.1. The conditional entropy is defined as: $H(B|A) = -\sum a_i p(a_i) \sum b_j p(b_j|a_i) \log_2 p(b_j|a_i)$. Intuitively, $H(B|A)$ gives the uncertainty with which we can predict the value of $B$ given that a value of $A$ appears [38].

We normalize the FRP-attribute matrix so that the entries of each column sum up to 1. For each FRP $a_i \in \mathcal{A}$, we define $p(a_i) = \frac{1}{|\mathcal{A}|}$. The normalized matrix for the running example is depicted in Table 6.4, in which each column stores the distribution $p(B|A = a_i)$.

Let us consider a particular clustering $C_k$ of the FRPs in $\mathcal{A}$. We introduce a third
random variable $C$ taking values from the set of clusters $\mathbb{C} = \{c_1, c_2, \ldots, c_k\}$. Mutual information measures the amount of information that two variables hold about each other. Intuitively, mutual information defines the amount of uncertainty (entropy) in one variable that is removed by knowledge of the value of the other. More precisely, we have: $I(B; C) = H(B) - H(B|C)$. Mutual information is symmetric, nonnegative, and equals 0 if and only if $B$ and $C$ are independent [38]. In our context, the mutual information $I(B; C)$ quantifies the information about the value of $B$ (domain attributes) provided by the identity of an FRP-cluster (a given value of $C$). The higher this quantify is, the more informative the cluster is about the attributes. Therefore, our goal is to choose $C_k$ in such a way that it maximizes the value of $I(B; C)$.

An agglomerative algorithm [3] was proposed to find a clustering of small cardinality and large information content. Similar to all agglomerative (bottom-up) techniques, the algorithm starts with the clustering $C_n$, in which each object $a_i \in A$ is a cluster by itself. As stated before, $I(A; B) = I(C_n; B)$. At step $n - l + 1$ of the algorithm, two clusters $c_x, c_y$ in $l$-clustering $C_l$ are merged into a single component $c^*$ to produce a new $(l - 1)$-clustering $C_{l-1}$. The clusters $c_x$ and $c_y$ to be merged are chosen to minimize the information loss in moving from clustering $C_l$ to clustering $C_{l-1}$. This information loss
is given by:  \( \delta I(c_x, c_y) = I(B; C_l) - I(B; C_{l-1}) \). We can also view \( \delta I \) as the increase in the uncertainty of predicting the domain attributes in the clusters before and after the merge. To complete the iterations in the agglomerative algorithm [3], the new component \( c^* = c_x \cup c_y \) has: \( p(c^*) = p(c_x) + p(c_y) \), and \( p(b_j|c^*) = \frac{p(c_x)}{p(c^*)} p(b_j|c_x) + \frac{p(c_y)}{p(c^*)} p(b_j|c_y) \).

Table 6.5 lists all pairwise values of \( \delta I \) in our running example. The value in position \((i,j)\) indicates the information loss we would incur if we chose to group the \(i\)th and the \(j\)th FRP together. Clearly, merging any pair of FRPs would lose some information, but merging FRP1 and FRP2 results in a minimal information loss. After each merge, we compute the probability of the newly formed cluster and the conditional probability of the attributes given the new cluster, according to the equations listed above. Then, at each iteration, we merge two clusters that incur the smallest value in \( \delta I \). The complexity of this algorithm is \( O(n^2 \log n) \), where \( n \) is the number of objects (FRPs) [3].

Figure 6.4 shows the dendrogram of the hierarchical clusters in our example, along with \( \delta I \) values indicating the information losses. The resulting clusters help make design decisions on system decomposition. For instance, our approach is able to discover both utility subsystems (e.g., searching module \{FRP7,FRP5\}) as well as functionally cohesive ones (e.g., reservation module \{FRP1,FRP2\}). Deciding a cutoff point requires domain knowledge as the threshold controls the amount of information loss in our summary of the set of primitive FRPs. A good choice for the granularity level is necessary to produce a concise and useful summarization of the requirements.

At this architectural design stage, a feature’s mandatory/optional property is likely to be uncovered. To further support the decision on variabilities, we can use the observation that each feature’s variability is pertinent to its parent [26]. In particular, when merging hierarchical clusters of a SPL’s sample applications [21], we compute the ratio of a feature’s occurrence to its parent feature’s occurrence. If this ratio is greater than a threshold chosen according to the domain knowledge, the child feature’s variability is set to be mandatory; otherwise, it is an optional feature [26]. The root of the merged
6.1.4 Clustering in Action: An Example

In this section, we study the order-processing systems in data-dominant domains. A data-dominant system is one that primarily concerns the integrity of the system data [26]. In this proof-of-concept example, for simplification, we chose two sample applications as if they were consolidated to form a SPL’s initial baseline. The first application is taken from [1], in which textual scenarios are provided for defining system functionalities. The second one is adapted from [143], in which use case descriptions are given for modeling requirements and deriving designs. Since these applications share similar business goals [143], we do not experience serious conceptual mismatches or terminological interferences [107]. Our samples are complementary rather than competing systems in the domain.

Our FRP-extraction algorithm [108] automatically generated about a dozen FRPs from each sample application’s textual requirements information. Following the semantic analysis described in Section 6.1.1, we were able to uncover 22 essential attributes in the feature tree is always mandatory.

Figure 6.4: Dendrogram of hierarchical clusters
domain, among which 7 were actors (including the ‘system-to-be’) and 15 were classes. As was mentioned, the domain experts must validate the objects to be clustered (extracted FRPs) and their attributes. In this exercise, the first author validated the extracted data against the original ones [1, 143]. The FRP-attribute matrix was determined manually but quickly. The whole data preparation and validation took approximately 10 minutes.

We now apply the on-demand cluster analysis framework to the consolidated data set. A partial result of user-centered clustering (cf. Section 6.1.2) is shown in Figure 6.5. Six identified features are presented with some FRPs belonging to more than one feature. The user also names the features by consulting the cluster centers. Of particular interest is the “data management” feature depicted with bold borders in Figure 6.5, which cuts across several other features. This indeed identifies an early aspect module related to persistence in data-dominant domains [131]. The clusters reduce complexity so that the users can prioritize the requirements more easily. For example, “catalog service” is a nice-to-have satisfier [126] in most stakeholders’ preference lists, so it is an optional feature in the order-processing SPL. “Customer service” has a rather conflicting ranking: the
customer support manager sees it as an essential dissatisfier [126], whereas clerks and accountants perceive it as a delighter [126]. This inconsistency can be resolved by treating “customer service” as an increment in the SPL’s blueprint [125]. Such an incremental (reactive) approach to SPL engineering is important since the assets extracted from sample applications are inevitably incomplete [35, 83].

We also perform design-driven clustering (cf. Section 6.1.3) based on the minimization of information loss when analyzing the FRPs extracted from the sample applications. Instead of showing the dendrogram of clusters, we present a domain feature diagram [75] in Figure 6.6. Essentially, a feature diagram is a hierarchy of common and variable features characterizing the set of instances of a concept [39]. In our case, the features of the concept “order-processing” are described, which is located at the top of the feature diagram. The solid edges connect a feature with its direct sub-features. The little circle defines the semantics of the edge: a filled circle means common (mandatory), and an outlined circle means variable (optional). In Figure 6.6, we make direct use of the overlapping clustering result to explicitly model the crosscutting feature “data management” and its sub-features. The crosscutting behavior is denoted by dotted arrows. This aspect module, which is built manually, shows one of the most salient features in data-dominant domains [26, 131]. The functional (top) part of Figure 6.6 is constructed by iteratively merging the features that result in a minimal information loss. The features and their variabilities were reviewed and validated by a domain expert. Such a clustering result, though inevitably incomplete, specifies an initial architectural baseline for the order-processing SPL.

We conduct a preliminary evaluation of the cost and the effect of the proposed approach. First, as the selected systems are of relatively small scale and the automated activities have polynomial time complexity, the construction effort is expected to be much less than manual effort. Remember that our goal is to complement, rather than replace, existing domain analysis techniques. One way to addressing the scalability issue is to
Chapter 6. Applications of FRPs in Product Line Engineering

introduce an intermediate step where we partition FRPs to meaningful subsets; however, we have not yet investigated this idea. Second, the features identified in our approach are compared with the ones presented in the original examples [1, 143] and in similar domains [26, 131]. The comparison shows a high precision (accuracy) but a limited recall (coverage), which is acceptable in extractive SPL development [83]. Third, even though there exist some metrics, measuring the quality of the generated clusters is subjective since stakeholders have different perspectives. Our framework takes various demands into account, and allows SPL features with different granularities to be browsed and recognized. Although the review of our clustering results by a domain expert is quite positive, in the longer term, we plan to carry out more in-depth empirical studies to determine the value of our approach.

Finally, we would like to share some observations and experiences from our study:
• It is important to realize stakeholder intentions and separate concerns in SPL requirements analysis. However, it is equally important to recognize the interplay of various factors and unite them into a coherent framework. In our study, for example, we are able to leverage the user-centered overlapping clustering result to modularize the crosscutting features in the architectural design.

• Each cluster represents a set of tightly-knit primitive functional requirements, so clusters (features) effectively raise the level of abstraction, reduce complexity, and facilitate communication in SPL engineering. Moreover, the grouped FRPs provide a context for stakeholders to compare and contrast their (abstraction) vocabularies [107]. In our study, for instance, the FRPs inside “data management” help build the correspondence between the non-functional requirements “persistence” and “integrity”.

• Even though clustering can be automated to a certain degree, domain knowledge is indispensable in SPL requirements analysis. In our framework, human intervention is needed to validate FRPs and their attributes, to examine threshold values (granularity levels), to determine feature variabilities, and to make sense of the resulting clusters. Tool support is desired for recording clustering criteria and design rationales during the entire process.

6.1.5 Related Work

Our work presented in this section (6.2) is related to several different efforts. John was among the first to recognize the importance of integrating legacy documentation assets into a SPL [70]. Basic elicitation techniques, patterns, and templates [71] are defined for extracting features, use cases, functional, non-functional requirements, project issues, usage constraints, etc. We focus on mining reusable functional primitives [100] from requirements documents, introduce a richer conceptual unit (FRP) to capture the
action-oriented concerns [147] in the domain, and develop an information retrieval technique [108] for FRPs’ extraction.

Likewise, our work is related to the effort of constructing feature models based on requirements clustering [26]. The objects to be clustered in their work are similar to the FRPs, but need to be identified manually. The relationships between the objects in [26] are also determined manually via feature and resource dependencies [86]. In contrast, we exploit the linguistic characterization of domain entities and functions, and provide automatic support for uncovering the homogeneous clustering objects (FRPs) and the semantic attributes. While we are keen to experiment the idea of using the occurrence ratio to decide a feature’s variability [26], we insist that domain knowledge should not be diminished.

Clustering is a mature research subject itself [61, 60], which has been employed in many disciplines, including software engineering. In Section 6.1.3, we adapt an information-theoretic clustering algorithm developed in reverse engineering [3] for requirements analysis. Clustering support for automated tracing is investigated in [44]. Some traceability techniques are based on machine learning or other supervised learning algorithms, which makes the quality of clusters highly dependent upon the training data. The clustering algorithms underlying our framework are intended to be unsupervised, which can be automated to a large extent.

Most requirements clustering techniques adopt some kind of agglomerative algorithm [60] to build hierarchical modules. Yauung [165] proposed a scenario-based prototyping technique to subdivide a software system into user-recognizable components where each component could be used, almost independently, to satisfy part of the user’s needs. We reinforce the idea of user-centered clustering in Section 6.1.2 to further support the SPL’s incremental delivery.

Li and colleagues [87] encapsulated requirements into clusters based on seven pre-fixed attributes: subject, object, action, functionality, quality, time, and location. Similar se-
mantic attributes, along with domain entities, are defined in [1], in which functional
decomposition and modularization are achieved by clustering requirements scenarios.
However, manual work is required to restore the verb-DO pair in each scenario description [1].

Our approach is advantageous over the above mentioned requirements clustering tech-
niques in that: 1) the semantic attributes are systematically uncovered on the basis of
Fillmore’s case theory [50] and contextual variability in goal-oriented RE [90]; 2) non-
 hierarchical (overlapping [27]) requirements clustering is first performed to explore the
crosscutting structures and behaviors.

6.1.6 Summary

Legacy systems and their documentation are valuable source for developing a SPL [70].
Despite the high quality single systems, the adaptability of the legacy assets can still
thwart a SPL’s adoption success [22]. One reason is due to the up-front cost and the level
of manual effort associated with traditional domain analysis methods. In this section,
we have presented a clustering framework for analyzing the functional requirements in a
SPL. We provide automatic support during the extracting and the clustering processes.
The on-demand analysis framework is, to the best of our knowledge, the first attempt
to take different stakeholder perspectives into account when discovering and organizing
SPL features. The proposed framework has addressed several important issues, such as
modularization and prioritization, in RE for SPLs.

Our work has also advanced the requirements clustering literature by examining clus-
ters that overlap and those causing a minimal information loss. We used a library MIS
example to illustrate our approach, and presented a more comprehensive study of order-
processing systems in data-dominant domains. These examples helped demonstrate the
applicability and usefulness, as well as evaluate the cost and benefit, of our approach.

The proposed approach is a work in progress, and therefore has a number of limitations
that we plan to investigate further. First, analysis of scalability and intermediate-level cluster generation is in order. Second, deciding the semantic attributes’ granularity level needs to be investigated. Meanwhile, we are open to integrating other attributes that are potentially useful to the SPL requirements clustering process. Such attributes may include finer-grained semantic roles, as well as feature dependencies and constraints. Third, we are excited to explore how different types of weighting schemes can affect, or ideally enhance the clustering results. Finally, we hope to incorporate the current framework into our repertory grid method for dealing with non-functional (quality) requirements [107]. We are certain that such a synthesis would help the stakeholders better identify, understand, and organize the SPL’s requirements assets.

### 6.2 Detecting Terminological Interferences

In SPL engineering, the original proposal of feature orientation envisaged that standard terms (features) used by the stakeholders would naturally emerge at the right level of abstraction [75], but many studies presented contradictory evidence. For instance, engineers working on the LG’s elevator control SPL did not agree on what specific features meant, even after 3 months of domain analysis [35]. Experience also showed that if two features simply had different marketing names, they were often mismatched and developed separately [40]. Mismatches in stakeholders’ vocabulary often occur, as an area of surprising controversy. When developing a SPL, experts would occasionally misunderstand one another, because they were using the same words in different ways. In fact, experts would sometimes be in “violent agreement” with one another, all the while expressing the same idea in different terms [35].

Reconciling the descriptions offered by different stakeholders is one of the hardest problems in RE [69]. People filter their observations of the world according to their interests, each person using his or her own conceptual framework. So, when we ask them
to describe something, their descriptions differ: they might focus on different aspects and choose different terms. In requirements analysis, stakeholders can have varied expertise areas, distinct responsibilities, and divergent goals. They might disagree over how to interpret phenomena in the problem domain, what the requirements are, and how to meet those requirements. As a result, they often express themselves using ambiguous or conflicting terms. For example, what one person calls “responsiveness” might correspond to “performance” in another’s description. For cell phone software, one stakeholder might interpret “usability” as “easy to learn”, another as “mobility”. Such mismatches in stakeholders’ vocabulary can be very hard to detect. We call this terminological interference.

Analysis of terminological interference is possible only if we can discover relationships between stakeholders’ mental models and the terms they use to describe them. George Kelly’s Personal Construct Theory (PCT) [78] addresses this issue. According to PCT, individuals develop their own set of mental constructs to help make sense of their environment. Researchers have used this theory to develop techniques for exploring personal constructs, most notably the Repertory Grid Technique (RGT) [52]. The RGT elicits personal constructs by asking people to compare and contrast objects in the domain of interest.

In this section, we show how to use the RGT, by leveraging the FRPs or other concrete entities like tasks, to tackle the terminological interference problem. We use the auto-marker SPL to illustrate our approach. We also conducted a pilot study for a nonprofit organization [107]. The studies show that our technique can readily identify agreements and mismatches in stakeholders’ terminologies, and that the technique fits into our framework’s lightweight philosophy in that it can be performed with proper tool support and without preliminary training or specific resources.
6.2.1 Terminological Interference

When people observe a complex problem domain, their observations are inevitably incomplete. Personal values and experiences act as a filter, leading them to focus on aspects that are particularly salient to them. This gives rise to many partial conceptual structures. When asked to articulate these, individuals choose terms that are personally meaningful. Often, people find it necessary to adapt or invent new terms to describe situations that they haven’t previously needed to articulate.

When stakeholders perceive a shared problem situation and attach terms to their concepts, four possible conditions exist for the relationship between their terminology and concepts, as summarized in Figure 6.7 (adapted from [145]). The challenge in knowledge elicitation is to discover which situation applies for a given set of stakeholder terms:

- **Consensus** is desirable; it gives stakeholders a basis for communication using shared concepts and terminologies.
Chapter 6. Applications of FRPs in Product Line Engineering

- *Correspondence* lays the ground for mutual understanding of differing terms through the availability of common concepts.

- *Conflict* can cause significant communication problems during RE activities.

- *Contrast* does not involve interference, strictly speaking. But the lack of shared concepts could make communication and understanding among stakeholders very difficult.

We interpret each correspondence and conflict as an instance of *terminological interference*; each can cause communication problems, if not identified and managed. On the other hand, we believe that terminological interference is both inevitable and useful in RE. It is inevitable because stakeholders have complementary perspectives and are unlikely to have agreed on a well-defined, shared terminology for describing the problem situation. It is useful because it provides an opportunity to probe differences in the stakeholders’ conceptual systems, challenge ill-defined terms, and identify new and productive distinctions for important concepts in the problem domain.

### 6.2.2 The Repertory Grid Technique

George Kelly’s Personal Construct Theory (PCT) [78] assumes that the meaning we attach to events or objects defines our subjective reality and thereby the way we interact with our environment. Kelly’s basic postulate of PCT is that a person’s thought processes are psychologically channelized by the ways in which he or she anticipates events. A key idea in PCT is the image of the person as scientist. According to the theory, each person constructs a model of the world (much as a scientist constructs a theory), acts on the basis of that model (as the scientist creates an experiment to test the theory), and then alters the model in the light of feedback from the results of his or her actions (as the scientist uses data from the experiment to modify the theory). This view shares much
of the spirit of the Inquiry Cycle [127], in which requirements models are theories about the world, and designs are tests of those theories.

A key message of PCT is that individuals set the measure of their own freedom and their own bondage by the level at which they choose to establish their convictions. Constructs are ways of construing the world, enabling people to respond to their experiences in ways that are explicitly formulated or implicitly acted out [78]. For example, the way in which I interact with my desk is determined by the way I construe it-do I polish it carefully because I see it as something to be looked after, or do I put my feet up on it because I see it as a convenient resting point? Thus, in Kelly’s theory, the notion of objectivity disappears, and the best we can do along these lines is intersubjectivity, thinking rather of a dimension representing degree of agreement between construers and degree of certainty of judgment.

PCT represents a coherent, comprehensive psychology of personality that has special relevance for psychotherapy. Researchers have developed PCT in conversational models of learning, using tools such as the Repertory Grid Technique (RGT) [52]. The RGT acts as an instrument for capturing the dimensions and structure of personal meaning, and provides a way for people to verbalize how they construe certain objects within an area of interest. These verbalizations are known as constructs, and the objects they refer to are called elements. A construct is a bipolar dimension, where each pole represents the extreme of a particular view or observation.

As an example, the area of interest might be how people construe certain information sources. In this example, the elements would be various information sources, such as TV, Radio, Newspaper, Newsgroup, so forth. A simple way to elicit a person’s constructs is to select a triad of elements and ask for a way in which two of them seem similar and how the third differs. For example, presented with the triad (A) TV, (B) Newspaper, and (C) Newsgroup, the person might say that A and B have many focuses, whereas C is singly focused. The construct ranging from “many focuses” to “single focus” can
be considered a rating scale using, for instance, a scale from 1 to 5. The person can now assign each element a rating on that construct. The same triad might elicit more constructs. For instance, the person might also say that B and C are text-based, while A delivers multimedia services. As more and more constructs are generated using different triads and the elements rated on them, a picture can be built up of an individual’s ways of construing the domain.

Figure 6.8 presents a sample repertory grid with constructs elicited in our current example. Each column represents an element from the domain and each row represents an elicited construct. Constructs are bipolar, so we label them using the terms that participants gave to describe the two poles during the elicitation. Each entry in the grid indicates how the participants rated the element in that column according to the construct in that row, using a five-point scale. By convention, “1” means we can best describe the element using the pole to the grid’s left, and “5” means it is best to use the pole to the grid’s right, with the remaining values indicating intermediate points on the scale.

Although the triad method is widely used in the RGT, there are many other ways to construct a repertory grid [52]. In our framework, we advocate a lightweight way of con-
structuring the repertory grid. For example, we could leverage the FRPs extracted from a natural language requirements document or the tasks extracted from a requirements goal model [105]. However, we believe the triad method can also be useful in eliciting constructs directly from stakeholders. Collected repertory grids are amenable to clustering analysis and many other measurements. Although most repertory grids are descriptive rather than evaluative in nature, the RGT does provide evidence of explanatory and predictive potential [52].

6.2.3 Managing Terminological Interference with Repertory Grid

A fundamental issue in applying the RGT is to decide the elements and the constructs. In our context, we treat FRPs or tasks as elements because the action-oriented concerns captured in these entities could establish a common ground among stakeholders. We use such elements to align SPL features [75] or non-functional requirements (NFRs) [30]. In this section, we use requirements goal models to illustrate our approach. In the next section, we show how our technique can be instantiated to cope with SPL features.

Goals express, at various levels of abstraction, stakeholders’ many objectives for the system under consideration. Goal-oriented RE uses goal models to elicit, elaborate, structure, specify, analyze, negotiate, document, and modify requirements [160]. Goal modeling shifts the emphasis in requirements analysis to the actors in an organization, their goals, and the interdependencies between those goals, rather than focusing on processes and objects. This helps us understand why a new system is needed and lets us effectively link software solutions to business needs. Requirements research has produced two principal goal-modeling techniques — Knowledge Acquisition in Automated Specification (KAOS) [42] and distributed intentionality (i∗) [168]. Both use goal models to provide criteria for determining whether requirements are relevant and complete.

Goal-modeling frameworks distinguish between hard goals – states that actors can attain – and softgoals, which can never be fully satisfied. System qualities such as relia-
bility, efficiency, and portability are typically expressed as softgoals to suggest that the intended software is expected to satisfy them within acceptable limits, rather than absolutely. Softgoals tend to express abstract concepts because they are difficult to express in a measurable way.

We therefore regard goals as personal constructs, to examine whether different stakeholders use the same terminology when describing their goals. In particular, we focus on softgoals, i.e., goals whose satisfaction cannot be established in a clear-cut sense. Softgoals are often hard to express in a measurable way, so ensuring that different stakeholders understand them in the same way is difficult.

To compare different stakeholders’ constructs, we must have an agreed set of elements. Requirements goal models typically describe tasks, which contribute in various ways to the satisfaction (or otherwise) of goals. Because tasks are much more concrete than softgoals, it is more likely that stakeholders will agree on their meanings: empirical evidence suggests that people are better at comprehending concrete RE concepts than abstract ones [124]. In particular, we assume that people focusing on similar topics can readily agree on the definition of a common set of concrete tasks within the area of interest. We then compare stakeholders’ softgoals by how they relate to this shared set of concrete tasks rather than by any terms the stakeholders use to describe them. Our approach involves four highly iterative and interactive activities: extraction, exchange, comparison, and assessment.

Extraction

A key RGT assumption is that a finite set of elements defines the context. We must carefully choose elements within the area of interest of the constructs we wish to study [52]. For instance, it bends our minds to consider “antique” or “modern” numbers and “prime” or “nonprime” furniture.

When analyzing goal models, we begin with some core agent or key activities in the
system; this generally provides a well-scoped area of interest. We carefully record each grid’s context so that we can perform sensible exchange and comparison.

We treat each softgoal in the context as a construct, identified as a pair of polar extremes corresponding to “make the goal” and “break the goal”. Then we select concrete entities, such as tasks related to the chosen constructs, as elements.

We then rate each element on each bipolar construct. For each grid, some ratings can be obtained from the goal models directly, some can be derived through label propagation algorithms [30], and the remainder needs to be completed by the stakeholder. We define a five-point scale to make such measures both subtle and specific:

<table>
<thead>
<tr>
<th></th>
<th>Break (strong negative)</th>
<th>Hurt (weak negative)</th>
<th>Neutral (unknown or don’t care)</th>
<th>Help (weak positive)</th>
<th>Make (strong positive)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Exchange**

Each grid expresses how a particular stakeholder views the domain and in what terms he or she makes sense of the underlying elements. In a shared context, each stakeholder’s personal construct system overlaps to some degree with others, and this lets people exchange their grid data to share their individual perceptions of the domain.

We exchange only concrete entities (that is, tasks) between stakeholders because at this stage, abstract constructs have meaning only within each person’s individual conceptual system. A construct is a discriminator, not a verbal label, so it is not transferable to another person without discussion and negotiation [52].

On the other hand, the concrete entities are exchanged, because to make comparisons across individuals and investigate construct similarity requires that each construes the same set of elements. This structural exchange keeps us from making assumptions about
the meanings of individuals’ constructs [105].

**Comparison**

We compare stakeholders’ softgoals according to how they array the set of common tasks in a particular context. We can examine any two constructs’ relationship by seeing to what extent one construct’s ratings of all the elements tend to match, or differ from, the other’s ratings.

If two softgoals relate to the tasks in the same or very similar way, we note them as a potential “correspondence”, even though they might be labeled differently. If two softgoals that are labeled using the same term relate to the tasks in a markedly dissimilar way, we note them as a potential “conflict”.

**Assessment**

The most important reason to analyze stakeholders’ goal models and compare their softgoals in a repertory grid is not the grid itself but the discussion that follows. Our approach is of practical value if our findings help identify the sources of existing interferences and then generate follow-up questions to resolve them.

**6.2.4 Pilot Study**

We carried out a pilot study to investigate the applicability of our RGT-based approach. The study context was a nonprofit organization, Kids Help Phone (KHP), which counsels kids and parents across Canada through the phone and Internet. Our pilot was part of a research project investigating the use of a goal-oriented framework for systematically analyzing the requirements for new Web-based counseling services [46].

At the project’s start, KHP project members conducted 14 stakeholder interviews, covering all major roles in the organization, and then used the interview transcripts to develop goal models. In the process of constructing these models, the team encountered
several problems emerging from diverse stakeholders with competing goals and differing vocabularies. For this reason, we considered the models appropriate candidates for applying the RGT.

Of particular interest for our study was a set of observations from the project’s earlier stages, when the team had to deal with terminological issues. The observations were as follows:

- Using each stakeholder’s own vocabulary, or a close paraphrase, helped to avoid modeling bias.

- Using more descriptive terms to name the model entities improved the model’s readability.

- Modelers who recorded where the terms came from and why they were chosen were able to build traceability on the fly.

- Modelers sometimes had to fabricate terms to express stakeholders’ requirements, and modelers differed in terminologies among themselves.

These observations led us to explore whether better approaches were available to handle terminological interference.

**Extraction and exchange**

We focused our study on the most important issue for KHP’s service planning: the counseling role itself. The data set contained interviews with three counselors, and the team had developed a separate goal model from each of these transcripts. Three KHP project members, whom we call Ana, Bob, and Cem, offered us these models and helped us extract model elements, complete repertory grids, and assess comparison results. Figure 6.9 shows the extracted entities from these goal models — 15 tasks and 12 softgoals.
In our approach, all the analysts share the set of tasks, which together determine the common ground. To circumvent bias, we asked a designated requirements engineer other than the original modelers to extract and consolidate the common set of tasks. The engineer extracted softgoals directly from each of the three models. The effort of extracting tasks and softgoals took about one hour.

Because we treat softgoals as personal constructs, we append the first letter of the stakeholder’s name to the name of each softgoal, so that those with the same label but different owners are treated as distinct constructs. So, strictly speaking, Figure 6.9 contains 20 softgoals, not 12. We then asked each analyst to rate all 15 tasks in his or her individual list of softgoals using the five-point scale defined earlier. They did so manually because of lack of tool support. In our study, each rating exercise lasted approximately 30 minutes.

<table>
<thead>
<tr>
<th>Tasks / Elements</th>
<th>Softgoals / Constructs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phone Counseling</td>
<td>Anonymous[Service] (A,B,C)</td>
</tr>
<tr>
<td>Web Service</td>
<td>Confidential[Service] (A,B)</td>
</tr>
<tr>
<td>E-mail Counseling</td>
<td>Safe[Service] (A,C)</td>
</tr>
<tr>
<td>Video Counseling</td>
<td>Immediate[Service] (B,C)</td>
</tr>
<tr>
<td>Text Messaging</td>
<td>Short[Waiting Times] (C)</td>
</tr>
<tr>
<td></td>
<td>Accessible[Response] (B)</td>
</tr>
<tr>
<td>Moderated Support Group</td>
<td>Interaction (B,C)</td>
</tr>
<tr>
<td>Moderated Discussion Board</td>
<td>Comfortablenesss (C)</td>
</tr>
<tr>
<td>Internal Communication</td>
<td>Improve[Counseling Skills] (A,C)</td>
</tr>
<tr>
<td>Public Speaking Workshops</td>
<td>High[Morale] (B)</td>
</tr>
<tr>
<td>Perform Standards Consistently</td>
<td>Make–Difficult[Work] (A)</td>
</tr>
<tr>
<td></td>
<td>Avoid[Burnout] (A,B)</td>
</tr>
</tbody>
</table>

Figure 6.9: List of all elements and constructs
Comparison and assessment

Using the FOCUS program [52], we analyzed the three repertory grids collected from Ana, Bob, and Cem after the extraction and exchange procedures and generated one resultant grid. FOCUS performs a two-way hierarchical clustering analysis and reorders the grid so that similarly rated elements are adjacent and similarly used constructs are adjacent. The full version of the resultant grid has 15 elements and 20 constructs, as Figure 6.9 indicates. Figure 6.10 shows a projection of the resultant grid with five tasks and six softgoals. FOCUS can also reverse constructs’ poles to show correlated values, so we have annotated the poles with “+” to denote the high end (5) of the scale and “−” to denote the low end (1) – that is, “breaking” a softgoal.

FOCUS builds *dendrograms* (tree diagrams, often used to illustrate how the clusters produced by a clustering algorithm are arranged) that illustrate the strength of association between elements and between constructs. For example, in Figure 6.10, the upper
dendrogram demonstrates the relationships between softgoals. To highlight the grid’s clusters, the ratings of 4 and 5 have dark shading, the ratings of 3 have light shading, and the ratings of 1 and 2 are unshaded. This helps the users of FOCUS easily identify blocks in the grid.

Figure 6.10 shows that the terms used to express softgoals interfered greatly. For instance, from Bob’s perspective, the softgoals Confidential and Anonymous were indistinguishable in terms of the tasks shown in the grid. If the stakeholders used them interchangeably, correspondence would be established; otherwise, we would need to distinguish the constructs through further elicitation. For example, we might ask Bob to specify a task that makes one softgoal but not the other. Although both Bob and Cem used the term “Anonymous”, they probably were not referring to the same concept. These two constructs were associated at the 70 percent level, one of the lowest matching scores between any two softgoals shown in Figure 6.10. We would flag this terminological inconsistency and explore it further.

The projected repertory grid shown in Figure 6.10 illustrates how we explore terminology problems. In our study, it would be inappropriate to deduce too much from the resultant grid. We would generate plausible hypotheses to be tested with follow-up discussion, rather than generating any firm conclusions about terminological interference.

To evaluate the results of our investigation, we presented preliminary findings to the requirements analysts Ana, Bob, and Cem, who created the initial goal models on the basis of the interview transcripts and have been in constant communication with the KHP organization. The precision of our approach was satisfactory: the analysts confirmed all the terminological interferences that we detected — five correspondences and three conflicts — in the context of counseling. However, assessing whether the approach can detect all occurrences of terminological interference in the goal models is harder. In this sense, the pilot trial resulted in only true positives. To validate the completeness of our approach, we must explore whether any softgoals that have interferences fail to manifest
in the repertory grid’s clustering analysis. We are designing further studies to investigate this.

**Observations and application to SPL engineering**

We can make several observations on the basis of our findings and discussions with the analysts. First, in some cases, correspondence between terms is easy and trivial to establish, so there is no need to resort to any complicated procedure. For example, all requirements analysts agreed that high-level softgoals about counseling such as Good, Helpful, Proper, and High-Quality had the same meaning, even though different stakeholders had adopted them. It would have been a waste of time to rate tasks on these softgoals, so we excluded them from our grid design.

Second, although statistical evidence shows high similarity between two terms, subtle and important differences might exist and must be investigated. When comparing Bob’s and Cem’s Anonymous[Service] softgoal on the basis of all 15 tasks in our study, we reached the similarity level at 86.7 percent, which was the second-highest match for each construct. If we treated this as a consensus, we would have missed an important distinction. In Cem’s opinion, the task E-mail Counseling contributed to Anonymity positively because people could protect their identities in the virtual space. But Bob thought E-mail Counseling could hurt Anonymity because most parents now install censorship software to protect their kids online, and this made the Anonymity of E-mail Counseling vulnerable. Exploring this difference yielded a more complete view of what Anonymous[Service] really meant to the counselors. This observation also indicates that using RGT numerical thresholds for establishing correspondence between constructs is unlikely to be useful in practice. In many cases, it is the subtleties that matter. Although our data supports quantitative analysis, qualitative inspection provides richer insights.

Third, conflicting relationships exist between mutually agreeable terms and concepts. For instance, both Ana and Bob agreed on the meanings of the task Consult New Tech-
nique and the softgoal Avoid[Burnout]. However, Ana perceived a negative relationship because Consult New Technique would MakeDifficult[Work] and therefore hurt the softgoal Avoid[Burnout]. From Bob’s standpoint, the task Consult New Technique could contribute positively to High[Morale], which helped to Avoid[Burnout]. This difference, which was captured in the grid, reflected conflict beyond the terminological level and sparked another discussion.

As we showed in Section 6.1 and pointed out earlier in this section, SPL features tend to express requirements at a higher abstraction level than primitive requirements [100]. Therefore, terminological interferences often occur when stakeholders describe the features for a SPL [35]. For example, we discovered in the auto-marker SPL that the “authentic commenting” feature in a professor’s description would correspond to what a TA featured “free-form grading”, both of which involved PFRs: highlight code segment, select pre-defined comments, optionally correct errors, and the like. We could employ the RGT approach described in this section to deal with such interferences among features. In this sense, the elements of the repertory grid would be the FRPs and the personal constructs of the grid would be features used in different stakeholders’ descriptions.

Similarly, we could use the extracted FRPs as the common ground to align NFRs
concerning a SPL. Figure 6.11 shows a sample repertory grid in the auto-marker SPL. Five extracted FRPs provide a shared context for comparing and contrasting the NFRs. The clustering analysis of this grid would flag the clash of “Usability” between the professor and the TA’s vocabularies due to the low similarity level at 70%. The work on definition hierarchies pointed out that it is acceptable for family members in a SPL to have different interpretations of the same NFR term [85]. We anticipate the mix-and-match of FRPs can help crystallize the varying NFR satisfying criteria; however, we have not yet investigated this idea.

Limitations

Although the KHP pilot study and the auto-marker SPL example demonstrated our approach’s usefulness, several limitations of the RGT became obvious. Most notably, we assume that stakeholders can mutually understand a common set of tasks or FRPs in a given context, but people might not be able to accurately interpret other people’s elements. Because the actual phrasing of elements might have a major impact on the proposed method, we have assigned a dedicated requirements engineer to consolidate the common ground for grid analysis. In the future, we plan to use a simple workshop to establish overlaps between stakeholders and combine the RGT and the nonfunctional requirements catalogue technique [30] to manage terminological interference.

The smooth establishment of an agreed set of tasks in our KHP study could be the result of comparing models that were essentially developed for different stakeholders fulfilling the same organizational role. To further investigate our approach’s scope of applicability, we plan to extend the interference analysis to heterogeneous stakeholder groups, such as people in different organizational roles or having different working experiences.

An earlier KHP case study found that serious scalability issues arose for goal modeling [46]. Our approach helped to address some scalability challenges by taking a prefixed element-construct view of models. The respondents from our pilot confirmed that ex-
tracting tasks and softgoals in the area of interest reduced the model’s complexity. They also pointed out that stakeholders could perform the proposed interference detection method without preliminary training or specific resources. However, some respondents commented that the effort of manually completing the extracted repertory grid was considerable and that the grid-rating process could have been more efficient and scalable. We are developing semi-automated tool support for grid generation via label-propagation algorithms [30], but it remains uncertain whether automatically filled partial grids can detect interferences. We shall clarify this issue through additional experiments.

Our pilot is a single study with one application rather than a controlled experiment on multiple representative subjects. We need more empirical evidence to strengthen the exploratory findings reported here. We must also address threats to validity—for example, does coincidence rating [52] lead to bias in the results? How many actual interferences does grid analysis fail to detect? To what extent can we generalize our approach to cope with requirements artifacts other than goal models?

6.2.5 Related Work

In [19], Boehm and In presented a knowledge-based tool that operates in the EasyWinWin framework [18] for analyzing conflicts among software quality attributes. A domain- or project-dependent taxonomy is required to serve as a stakeholder checklist for requirements negotiations.

If a particular softgoal is considered in the EasyWinWin framework, an explicit definition must be provided \textit{a priori}. Then, stakeholders suggest modifications to the taxonomy of topics to reflect their own interests. If terminological conflict occurs, i.e., stakeholders use the same terminology for different concepts, an issue is recorded and options to resolve the conflict are sought. At the end of each negotiation round, a consistent taxonomy or an agreed vocabulary must be achieved. In other words, all stakeholders need to agree on a globally consistent set of terms in order for the EasyWinWin framework
to proceed. The method emphasizes collaboration, and the success of the negotiation process is highly dependent on a “facilitator”. The key for the facilitator is to accurately identify (terminological) conflicts, strategically propose options to manage the conflicts, and effectively facilitate the process to reach an agreement.

This is an ad hoc method for addressing terminology problems in RE. It focuses only on terminological conflict, and no attempt is made to systematically identify overlap and build correspondence between terms used among stakeholders. Requiring global consensus is another limitation, as this can force stakeholders to reach an invalid consensus. As Easterbrook [45] pointed out, in many cases, negotiating terminological differences is a waste of time, and participants should agree to differ. It may be sufficient just for the participants to be aware of such conflicts.

Natural language processing (NLP) is expected to play an essential role in handling terminological interference in RE. However, it is argued by Ryan [138] that the potential role of NLP in the RE process has been overstated in the past. It is suggested that the validation of requirements must remain an informal, social process [138].

Dag et al. [116] followed a standard linguistic engineering process (flatten, tokenize, stem, and remove stop words) to calculate the terms’ similarities by vector-space models and the cosine measurement. This method, like most NLP-based RE approaches, assumes that the requirements documents are based on semi-formal models, or at least structured English. This assumption limits the linguistic elements used in requirements models to be a subset of the natural language. Typical restrictions include: (i) grammar – aiming to have syntactic constructions that are easier to analyze by requiring, for example, shorter phrases, using the active voice, by avoiding anaphoric references, etc.; and (ii) vocabulary – aiming to reduce ambiguity of terms [92].

However, to restrict the grammar or vocabulary of linguistic elements used in requirements models is actually to force the user, or even the analyst, to express what the models permit to be represented, rather than the real requirements of the system. So, a
Chapter 6. Applications of FRPs in Product Line Engineering

<table>
<thead>
<tr>
<th>Conventions</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>3. Noun[Topic] (that needs no adjective to clarify the meaning)</td>
<td>Quality[Topic]  TrustWorthiness[Topic]</td>
</tr>
</tbody>
</table>

Figure 6.12: Some naming conventions proposed for $i^*$ modeling framework

customer-oriented NLP-based approach is necessary [92].

An interesting project has been conducted [31] in defining naming conventions to consistently and meaningfully label $i^*$ entities. Naming conventions are the rules created to govern the naming of things. By adopting certain naming conventions, we can have appropriate, consistent, and meaningful names for elements in requirements models so that the maintenance effort is reduced, the readability is improved, and the understandability is increased. The naming conventions of softgoals and some examples are given in Figure 6.12. In practice, the names of softgoals used in the literature ignore such conventions. For example, the softgoal “Keep Private Information Confidential” used in [167] could easily be misinterpreted as a task. Adoption of consistent and succinct naming conventions is good practice in Computer Science, but the project of defining naming conventions for entities in $i^*$ models is in its early stage. More empirical studies are needed to investigate the usefulness and the scope of applicability of naming conventions.

Although the RGT was originally developed in the context of clinical psychology, it
has long been recognized as a domain-independent method for externalizing individuals’ personal constructs, and researchers and practitioners in diverse fields such as psychology, education, business management, and so on have applied it in a wide variety of situations far removed from clinical psychology. As far as the RGT’s applications in RE, Shaw and Gaines developed one of the initial approaches to repertory grid requirements elicitation. They based the approach on their knowledge engineering work and introduced George Kelly’s Personal Construct Theory [78] as a universal foundation for modeling methodologies [146].

Maiden and Rugg specify a situation where the RGT fits for requirements acquisition: package selection. Purchasing a software package often involves selection, and in such cases, requirements for the new system should act as selection criteria. If candidate packages are known, the RGT becomes an effective acquisition method because it explicitly encourages respondents to give criteria that discriminate between elements such as software packages [94].

Hassenzahl and Wessler explore the RGT’s practical value in gathering design-relevant information about early artifact prototypes designed in parallel. Personal constructs (for example, boring-interesting, graspable-abstract) that people employ when confronted with design alternatives play an important role in narrowing down the design process [62].

Delugach and Lampkin adopt the RGT’s triad method to elicit and classify requirements, distinguish and measure correlations between requirements, and reveal system qualities [43]. They use requirements as elements in their work, so that the elicited constructs show how stakeholders construe these requirements. However, the ability to compare stakeholders’ constructs depends on agreeing to a well-defined set of elements first. As we can never be sure that two stakeholders understand a particular requirement in the same way, it is unclear how useful it is to elicit personal constructs with respect to the requirements themselves. Recent work has taken a finer-grained look at requirements goal models by treating softgoals as personal constructs so as to discover early
aspects [105] and to identify softgoal contributions [57].

6.2.6 Summary

Problems with stakeholder terminology are endemic in RE. As our study shows, stakeholders do not use their terminologies consistently when stating requirements, including when doing SPL engineering. However, these terminological problems need not be a barrier to understanding stakeholders’ goals and requirements, if approached carefully. An exploration of how different stakeholders apply these terms can be extremely useful for a deeper understanding important concepts in the problem domain. One of the strengths of the RGT is that it avoids the problem of imposing an (unnatural) terminology on stakeholders – the meaning of a term is essentially treated as a relationship between signs and actions.

The ideas of personal construct theory lead to an analysis of terminological interference in which the aim is to gain a deeper understanding of how stakeholders construe the problem domain. The results of such an analysis are informative but not judgmental. Hence, we do not address whether any particular use of the terminology is “right” or “wrong”, or even whether a particular definition of a term is better than another. Most RE activities take place without correct and consistent models, and so, as indicated by Feather [49], hinge on getting right from wrong. We expect our proposed approach to act as a helpful initial step toward a more comprehensive framework for thoroughly understanding and adequately reflecting stakeholders’ desires and needs.

Further research includes the investigation of techniques for merging individual repertory grids to construct well-categorized taxonomies to organize large numbers of artifacts in RE, especially in RE for SPLs. According to our approach, such a taxonomy can be viewed as a catalogue of how constructs represented by linguistic symbols relate formally in a particular context. We also plan to design further experiments to test our hypothesis that concrete entities like FRPs or tasks provide a core set of common ele-
ments that a group of participants can all meaningfully construe. This is critical to all RGT-based approaches, and can lead discussion to exploring the ongoing debate about “whether elements exist independently of constructs, or whether in fact elements are also constructs” [97].
Chapter 7

Conclusions

In this chapter, we summarize the contributions made in this thesis and outline several directions for future research.

7.1 Summary

SPL engineering has the potential to offer practitioners order-of-magnitude improvements in software quality and productivity. It is argued that it takes two or three products’ delivery time for an organization to reap the benefit (in terms of development and maintenance cost) by adopting SPL engineering over one-of-a-kind system development [35]. Two or three is a very good number because it is hard to imagine a product line without at least two or three family members. The assumption is, of course, that one must build the right core assets and build them right.

Contemporary SPL methods often adopt the proactive model, attempting to build a relatively complete and stable (5-6 years [84]) set of core assets for reuse. In practice, the substantial up-front effort and the abrupt transition from existing practices associated with the proactive model present a prohibitive SPL adoption barrier for many organizations that could otherwise benefit. In other words, it is practically impossible to build the perfect assets for the first time [122]. A more feasible and economical SPL adoption
strategy is a combination of the extractive and the reactive models, in which we reuse existing products for the SPL’s initial baseline and then co-evolve the core assets and the products [83].

In this thesis, we have presented a framework for applying lightweight techniques to extract, model, and analyze a SPL’s requirements assets. By lightweight, we indicate that the methods employed in our framework are striving for increased automation and greater generality, and that the methods can be used to perform partial analysis without a commitment to developing a complete specification of the domain.

A fundamental contribution of this thesis is the introduction of FRPs (functional requirements profiles). In Chapter 3, we defined FRPs as a way to capture the domain’s action themes, and further contributed a semi-automated technique for extracting the FRPs from a natural language document. Our work substantiated the extractive SPL adoption model in dealing with the requirements assets.

In Chapter 4, we modeled the extracted FRPs by analyzing their semantic cases and by extending the OVM (orthogonal variability model) notation [125]. We contributed a set of heuristic rules for uncovering the variation dimensions and dependencies. We also discussed merging FRP-based OVMs and handling variability in the merge process. Our treating of FRPs as variation points offered practical guidelines for adopting the OVM framework.

In Chapter 5, our framework took quality requirements into account. We contributed a novel way to connect functional profiles and quality requirements via SEI’s quality attribute scenarios [10] and formal concept analysis [54]. We showed that, by manipulating the concept lattice, we could effectively manage requirements interactions [135] in the presence of an evolving SPL.

Two applications of FRPs were presented in Chapter 6. We used the FRPs and their essential semantic cases to perform on-demand clustering by recognizing stakeholders’ different goals in the analysis. We also advanced the requirements clustering literature
by examining clusters that overlap and those causing a minimal information loss. To detect terminological mismatches and interferences in software development, we used the FRPs to establish the context and applied ideas from psychology [78].

We have conducted several empirical studies to evaluate our framework: a pilot study on an auto-marker SPL, a case study with IBI Group’s Toronto office on a family of traffic management systems, a case study on a mobile game SPL, a case study with a social service organization in detecting terminological interferences in requirements models, and a couple of proof-of-concept examples on a media-shop software and a library MIS product line. These studies were reported in different chapters throughout the thesis.

The framework of a coherent set of lightweight techniques, together with the empirical evaluations, presented in this thesis filled the void with respect to extracting a SPL’s requirements assets. Other than our framework, Alves et al. [2] presented another framework in extractive SPL RE. It has three steps. First, use IR techniques to calculate a similarity matrix of one application’s requirements documents. Second, perform hierarchical clustering to obtain the feature model for each application. Third, merge, or rather do a disjoint union, of the applications’ feature models to obtain a feature model for the domain. Although their idea is similar to ours, distinctions exist. First, we cluster FRPs (cf. Chapter 6) rather than requirements sentences. This makes sure the objects to be clustered are at relatively the same abstraction level, making the clustering results more sensible. Second, our framework allows the stakeholders to plug in different clustering algorithms depending on their goals. Third, since disjoint union is only a special case of merging, our framework deals with model merging more generally. We look forward to the synthesis of emerging frameworks in extractive SPL RE.

We want to emphasize that our framework is intended to complement, not to replace, existing RE approaches for SPLs. We believe that the set of low adoption threshold techniques introduced in this thesis could lower the SPL adoption barrier, and more importantly, could act as a critical enabler for practitioners to capitalize on the order-of-
Chapter 7. Conclusions

magnitude improvements offered by SPL engineering.

7.2 Future Work

In this section, we outline some of the future research directions.

7.2.1 Tool Support and Empirical Studies

The empirical studies we conducted revealed some limitations that we plan to address in the near future. Whenever possible, we expect to provide automatic tool support for the users. The increased automation is in accordance with our philosophy of lightweightness. One of the immediate enhancements we plan to implement is domain concept identification. The current FRP-extraction method depends on the SRS’s definition section to recognize the domain concepts, yet we found out that this source is far from sufficient in practice. In the IBI case study, for instance, we used educated guess to recognize the concepts “response time” and “travel time”. While having domain experts validate the FRPs would offer a complete solution, some off-the-shelf tools, especially those based on lexical analysis, could help ameliorate the problem. Xtract [148] is such a tool. It can produce the biggest possible $n$-word lexical affinities ($n > 2$) from two-word associations. The evaluation of Xtract on a corpus of stock market news reports resulted in an estimated precision of 80% [148]. The underlying sliding window technique used in Xtract is essentially the same as the one used in the current FRP-extraction algorithm, making it straightforward for tool integration.

It is important to note that tool support is only a means to an end, not the end itself. In some occasions, tool building may be trivial. For example, we suspect that the SRS’s TOC, headings, and subheadings would be very useful clues for identifying, or confirming the identified, FRPs. Although a standard word processor should be a good enough tool support for such a task, it is more crucial to test the underlying hypothesis empirically,
e.g., through a controlled experiment. We plan to formulate a set of plausible hypotheses on the basis of our exploratory IBI case study, and design further empirical studies to test these hypotheses.

### 7.2.2 Tracing Stakeholder Concerns

The framework proposed in this thesis deals primarily with requirements. A natural extension would be to trace the stakeholder concerns identified in the requirements phase throughout the software life cycle. We feel that our approach has a strong connection with the natural language based program analysis [147], in which the problem-domain action-oriented concerns are effectively extracted from code comments and method signatures. Establishing traceability links is critical for validating stakeholder concerns during software maintenance and evolution. More sophisticated techniques such as LSA (latent semantic analysis) showed promising results for generating candidate links to trace requirements concerns [63].

We pointed out in our case studies that the extracted assets are under-specified because the extraction sources cover the domain only partially. Reactive development aims to enrich the asset base by embracing change and making the SPL’s core assets and the individual products co-evolve [83]. Tracing concerns in this context means updating the asset base continuously, and visualizing and analyzing the change impacts. Our work on concept analysis [110] has shed some light in this direction.

### 7.2.3 Model Management

The OVMs we currently build focus on system functionalities. This represents only one of the many modeling focuses. In order to understand and communicate the stakeholders’ goals and needs, we may build models of a system-to-be or of a SPL to reason about the structural, behavioral, and functional properties. With the increasing globalization of single-software and SPL development, modelling has evolved from a centrally-managed
activity into a distributed endeavor that may involve multiple teams spread across multiple sites. These teams typically build multiple inter-related models representing different perspectives and development concerns. In our scenario, several OVMs may be constructed based on multiple sources (SRS’s). Keeping track of the relationships between these models as they evolve, managing consistency, and constructing a global view of the models are major challenges.

Recent work on model management (e.g., [21, 104]) has focused on defining a set of operators for manipulating partial, distributed, and inconsistent models. In our current work, we explored the OVMs’ interrelationships mainly from the terminological perspective. In the future, we plan to take advantage of the faceted classification [129] to investigate the relationship between FRP-based OVMs, since both FRPs and faceted descriptors are textual and an FRP’s semantic cases are analogous to a descriptor’s facets. In addition, we are keen to discover models’ interconnections from structural and behavioral points of view.

7.2.4 Software Modularity

It has long been recognized that modular systems are easier to produce, maintain, and evolve [119]. However, complex problems like SPLs are hard to decompose cleanly, and any choice of decomposition will inevitably give rise to concerns that are tangled or scattered in the resulting structure. Aspects provide richer notions of modularity by explicitly considering the concerns that cut across multiple system components [102]. Although much focus to date is on programming mechanisms [79], researchers have begun to raise the levels of abstraction and separate concerns in requirements and design.

Aspects recently find their applications in RE for SPLs. It is argued that the effectiveness of a SPL approach directly depends on how well feature variability within the portfolio is managed from early analysis to implementation and through maintenance and evolution [125]. Variability of features often has widespread impact on multiple arte-
facts in multiple lifecycle stages, making it a predominant engineering challenge in SPL engineering. We plan to adopt the notion of aspect to address such a challenge. This line of research would yield an improved modularization of variations, their holistic treatment across the software lifecycle activities, and advanced maintenance of their (forward and backward) traceability.
Bibliography


[40] James C. Dager. “Cummin’s Experience in Developing a Software Product Line Architecture for Real-time Embedded Diesel Engine Controls”. In \textit{Proceedings


Bibliography


the 11th IEEE International Requirements Engineering Conference (RE’03), pages 117–126, Monterey Bay, California, USA, September 2003.


[74] Hermann Kaindl, Sjaak Brinkkemper, Janis A. Bubenko Jr., Barbara Farbey, Sol J. Greenspan, Constance L. Heitmeyer, Julio Cesar Sampaio do Prado Leite, Nancy R.


International Requirements Engineering Conference (RE’06), pages 76–85, Minneapolis, Minnesota, USA, September 2006.


Conference on Artificial Intelligence (AAAI’04), pages 1024–1025, San Jose, California, USA, July 2004.


Appendix A

Questionnaire

Evaluation of FRPs (Functional Requirements Profiles) and OVMs (Orthogonal Variability Models)

Question 1 (Scope): Do you think the model elements (FRPs and semantic cases) capture important domain elements? Are the results surprising? Insightful?

Question 2 (Products’ Similarity): Do you find using the FRP-differences for assessing products’ similarity is sensible? Promising?
Question 3 (OVMs): Do you think the commonality, variability, and dependency captured in the OVMs are accurate? Insightful?

Question 4: According to your experience, do you think that this approach (FRPs + OVMs) provides sufficient constructs and guidelines to be tested on a limited basis before adoption?

(Triability is the degree by which the product can be tried on a limited basis before adoption.)

Question 5: Do you see preliminary observable results from the application of the proposed approach to extracting and modeling a domain’s requirements assets?

(Observability refers to the observable results deriving from the use of the new product.)
Question 6: Compared to relevant techniques you are aware of, do you think that the adoption of the proposed approach can better help you improve the quality of requirements engineering (elicitation, analysis, documentation, etc.) for a software product line?

(Relative advantage is the perception of how much better the innovation is than the competing solutions currently adopted.)

Question 7: Do you think that the proposed approach is overly complex to be understood and used?

(Complexity refers to the fact that the innovative product should not be overly complex to understand and to use.)

Question 8: Do you perceive the proposed approach to be compatible and consistent with the existing practices, values, standards, and technologies shared in your organization?

(Compatibility measures how the innovation is perceived as compatible and consistent with existing practices shared among the community users.)