Concept Analysis for Product Line Requirements

Nan Niu
Department of Computer Science
University of Toronto
Toronto, Ontario, Canada M5S 3G4
nn@cs.toronto.edu

Steve Easterbrook
Department of Computer Science
University of Toronto
Toronto, Ontario, Canada M5S 3G4
sme@cs.toronto.edu

ABSTRACT

Traditional methods characterize a software product line's requirements using either functional or quality criteria. This appears to be inadequate to assess modularity, detect interferences, and analyze trade-offs. We take advantage of both symmetric and asymmetric views of aspects, and perform formal concept analysis to examine the functional and quality requirements of an evolving product line. The resulting concept lattice provides a rich notion which allows remarkable insights into the modularity and interactions of requirements. We formulate a number of problems that aspect-oriented product line requirements engineering should address, and present our solutions according to the concept lattice. We describe a case study applying our approach to analyze a mobile game product line's requirements, and review lessons learned.

Categories and Subject Descriptors

D.2.1 [**Software Engineering**]: Requirements/Specifications; D.2.10 [**Software Engineering**]: Design—*Representation*

General Terms

Design, Documentation

Keywords

Functional requirements profiles, quality attribute scenarios, product line engineering, formal concept analysis

1. INTRODUCTION

Software product line (SPL) engineering [13, 35] is one of the success stories of software reuse whose purpose is to improve quality and productivity. Although much of the SPL research to date has focused on code reuse, 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 [4]. Not only was reuse identified early, but also the effectiveness of reuse was enhanced as the engineer can work on the abstractions closer to the system's initial concepts.

Requirements abstractions are decomposed functionally in most contemporary SPL approaches like FODA (feature-oriented domain analysis) [18] and PRS (product-line requirements specification) [14], since system functionality represents the very noticeable aspect of a feature [18]. 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 [23] 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 [37]. The role of aspects in modeling SPL variabilities was examined in [25]. 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 [22]. 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.

In our earlier work, we developed a semi-automated technique for extracting a SPL's functional requirements profiles (FRPs) from natural language documents [29]. FRPs, which capture the domain's action themes at a primitive level, are identified on the basis of lexical affinities that bear a verb-DO (direct object) relation. We also analyzed the essential semantic cases associated with each FRP in order to model SPL variabilities [29] and uncover early aspects [30]. However, identifying aspects was achieved by organizing FRPs into overlapping clusters [30] without explicitly considering quality requirements.

In this paper, we investigate both functional and quality requirements via concept analysis [15]. Our goal is to efficiently capture and evolve a SPL's assets so as to gain insights into requirements modularity. To that end, we first set the context by leveraging FRPs and the SEI's quality

AOSD'09, March 2–6, 2009, Charlottesville, Virginia, USA.

attribute scenarios [5]. 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 that aspect-oriented 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 our work 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.

2. PRELIMINARIES

2.1 RE for SPLs

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 (RE) – change management" process [13]. Most SPL approaches therefore advocate reusing RE artifacts, as well as processes. Key RE activities include: plan and elicit, model and analyze, communicate and agree, and realize and evolve requirements [33]. In addition, SCV (scope, commonality, variability) analysis is a central theme running throughout a SPL's life cycle [47].

Domain analysis [36] is often used to identify requirements of a class of similar systems. An underlying premise of domain analysis is that *proactively* building a domain model pays off in subsequent application engineering. In practice, the up-front cost and the level of manual effort associated with proactive domain analysis present a prohibitive SPL adoption barrier for many organizations that could otherwise benefit. Krueger addressed this problem by showing that building a SPL *extractively* and *reactively* was more economical [22]. The tenets of these flexible models 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.

The results from SPL RE include the scope of the domain, domain vocabulary and concepts, and feature models describing the commonalities and variabilities of domain concepts and their interdependencies [47]. The use of SPL "features" is motivated by realizing that customers and developers communicate requirements in terms of "features the product has and/or delivers" [18]. Features, then, are distinctively identifiable functional abstractions that must be implemented, tested, delivered, and maintained [19]. This view pays little attention to quality requirements [1], such as reliability and usability.

Quality requirements not only describe how well system functions are accomplished, but also represent global concerns that are natural to be implemented as aspects [21]. 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 [23]. Our work takes both quality and functional requirements into account. Next, we show how we identify functional requirements primitives and make use of quality attribute information.

2.2 FRPs and Quality Attribute Scenarios

The benefits of getting quality requirements "right" are substantive in SPL engineering [9], 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 [21] 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 [28], even though several standards (e.g., ISO/IEC 25030) exist [8]. 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 [28]. We further contributed an information retrieval technique to effectively extract the FRPs from existing requirements documents [29]. FRPs, consisting of verb-DO (direct object) pairs, characterize product functionalities and action-oriented concerns in the domain. Sample FRPs for the media shop domain [27] are "navigate shop" and "customize toolbox". Note that many RE methods have implicitly used the verb-DO pair to model the functional unit: a use case, a primitive requirement [26], a goal concern (task) [24], to name a few. Our work has made this strategy explicit and operational in RE. 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 [28]. To bridge the gap, we take

[1]We have chosen to use the umbrella term "quality requirements" [8] in this paper to mean requirements that describe desired system qualities, which are also known as nonfunctional requirements, softgoals, and quality attributes in the literature [33]. This chosen terminology shall not be confused with high-quality requirements.

MEDIA SHOP	free−distribution	timely	paper	sound
CD	X			X
MAGAZINE		X	X	
NEWSPAPER	X	X	X	
VIDEOTAPE				X
BOOK			X	

(a) Formal context.

$\top$	({CD, MAGAZINE, NEWSPAPER, VIDEOTAPE, BOOK}, Φ)
c_1	({CD, VIDEOTAPE}, {sound})
c_2	({CD, NEWSPAPER}, {free−distribution})
c_3	({MAGAZINE, NEWSPAPER, BOOK}, {paper})
c_4	({MAGAZINE, NEWSPAPER}, {timely, paper})
c_5	({NEWSPAPER}, {free−distribution, timely, paper})
c_6	({CD}, {free−distribution, sound})
$\bot$	(Φ, {free−distribution, timely, paper, sound})

(b) Concepts for the formal context.

Figure 1: Binary relation.

advantage of the SEI's quality attribute scenarios [5]. 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 [2] provides an operational definition for some quality attributes [5]. 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 [20], our work uses them to investigate the relationship and modularity of a SPL's requirements.

2.3 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 [7] in 1940. For more detailed information on FCA, we refer to [15], where the mathematical foundation is explored. FCA deals with a relation $\mathcal{I} \subseteq \mathcal{O} \times \mathcal{A}$ between a set of objects $\mathcal{O}$ and a set of attributes $\mathcal{A}$. The tuple $C = (\mathcal{O}, \mathcal{A}, \mathcal{I})$ is called a *formal context*. For a set of objects $O \subseteq \mathcal{O}$, the set of common attributes $\sigma(O)$ is defined as: $\sigma(O) = \{a \in A \mid (o, a) \in \mathcal{I} \text{ for all } o \in O\}$. Analogously, the set of common objects $\tau(A)$ for a set of attributes $A \subseteq \mathcal{A}$ is defined as: $\tau(A) = \{o \in O \mid (o, a) \in \mathcal{I} \text{ for all } a \in A\}$.

A formal context can be represented by a relation table $\mathcal{R}$, where rows represent the objects and columns represent the attributes. An object o_i and attribute a_j are in the relation $\mathcal{I}$ if and only if the cell of $\mathcal{R}$ at row i and column j is marked by "×". As an example related to the media shop, a binary relation between a set of objects {CD, MAGAZINE, NEWSPAPER, VIDEOTAPE, BOOK} and a set of attributes {free-distribution, timely, paper, sound} is shown in Fig. 1a. For that formal context, we have: $\sigma(\{\text{CD}\}) = \{\text{free-distribution, sound}\}$, and $\tau(\{\text{timely, paper}\}) = \{\text{MAGAZINE, NEWSPAPER}\}$.

[2]For the rest of the paper, "scenario" refers to the quality attribute scenario, unless otherwise noted.

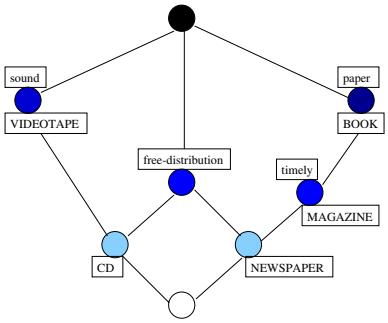


Figure 2: Concept lattice in sparse representation.

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 $extent(c)$, and A is called the *intent* of c, denoted by $intent(c)$. In Fig. 1b, all concepts for the relation in Fig. 1a are listed. The set of all concepts of a given formal context forms a partial order via the subconcept-superconcept ordering: $c_1 \leq c_2 \Leftrightarrow extent(c_1) \subseteq extent(c_2)$, or dually, with $c_1 \leq c_2 \Leftrightarrow intent(c_1) \supseteq intent(c_2)$. 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 Fig. 1b, we have $c_5 \leq c_3$.

The set $\mathcal{L}$ of all concepts of a given formal context and the partial order $\leq$ form a complete lattice, called *concept lattice*: $\mathcal{L}(C) = \{(O, A) \in 2^{\mathcal{O}} \times 2^{\mathcal{A}} \mid A = \sigma(O) \text{ and } O = \tau(A)\}$. The *infimum* ($\sqcap$) of two concepts in this lattice is computed by intersecting their extents, and the *supremum* ($\sqcup$) is determined by intersecting the intents. The infimum describes a set of common attributes of two sets of objects, whereas 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 Fig. 1a can be depicted as a directed acyclic graph whose nodes represent the concepts and whose edges denote the relation $\leq$ as shown in Fig. 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. For example, consider the node labeled above by "timely" and below by "MAGAZINE". This node represents the concept ({MAGAZINE, NEWSPAPER}, {timely, paper}), i.e., c_4 in Fig. 1b.

2.4 Running Example

We use the media shop [27] in the e-commerce domain to illustrate our approach; a fuller case study is presented in Section 5. 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:

Table 1: Extracted Requirements Constructs

Functional Requirements Profiles	Quality Requirements
FRP_1: navigate shop	Q_1: +U (positively contribute to usability)
FRP_2: search product	Q_2: +A (positively contribute to accessibility)
FRP_3: customize toolbox	Q_3: −M (negatively contribute to maintainability)
FRP_4: select language	
FRP_5: monitor quantity	
FRP_6: generate report	
FRP_7: create account	

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 [20], so we characterize system uses from various views. Second, although some scenario formatting template (e.g., stimulus, artifact, response) is proposed [5], we tend not to impose how a scenario is structured, but support narratives in the general sense [20]. Third, each scenario necessarily contains some action-oriented concerns, and by definition, each scenario provides quality attribute information. Table 1 lists the FRPs and the quality requirements considered in the scenarios. Finally, 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 [22] 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.

3. 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 [42]. 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 quality 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 [20].

Table 2: Crosscutting Relations

	FRP							Q		
	1	2	3	4	5	6	7	1	2	3
Sce_1	×	×	×	×				×		×
Sce_2	×		×		×	×		×	×	
Sce_3	×	×			×		×		×	×

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 Sce_1 and Sce_2, and both usability and maintenance are tangled in Sce_1. The same can be said for many functional concerns. Table 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 1, we use "+" (resp. "−") to denote a quality requirement is positively (reps. negatively) affected, though finer scales of measuring contributions [27] 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 2. We exploit scenarios to bridge this gap. In particular, we instantiate formal context by setting the objects ($\mathcal{O}$) to be the FRPs and the attributes ($\mathcal{A}$) to be the scenarios. The binary relation ($\mathcal{I}$) is given by the inclusion relation, as illustrated in Table 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 INS_1 ($\mathcal{O}$ = FRPs, $\mathcal{A}$ = scenarios), in which mapping the functional units to objects seems natural. However, three other instantiations of formal context exists:

- INS_2 ($\mathcal{O}$ = quality requirements, $\mathcal{A}$ = scenarios);
- INS_3 ($\mathcal{O}$ = scenarios, $\mathcal{A}$ = FRPs); and
- INS_4 ($\mathcal{O}$ = scenarios, $\mathcal{A}$ = quality requirements).

If we adopt INS_2, we would employ the same strategy of examining the concept lattice by combining scenarios in order to uncover requirements interactions. INS_3 and INS_4 are isomorphic to INS_1 and INS_2, respectively, due to the duality principle of lattice theory [7]. 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 [15]. 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 INS_1.

The concept lattice of our running example is shown in Fig. 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

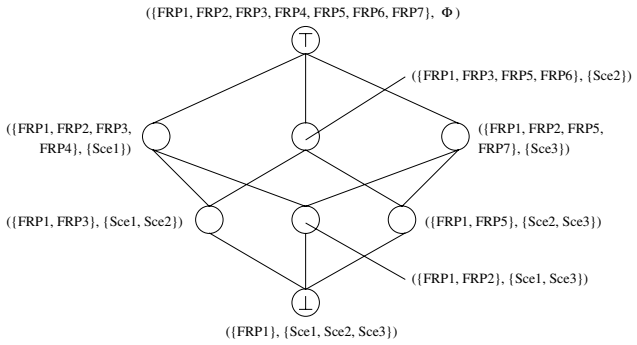


Figure 3: Concept lattice for the running example.

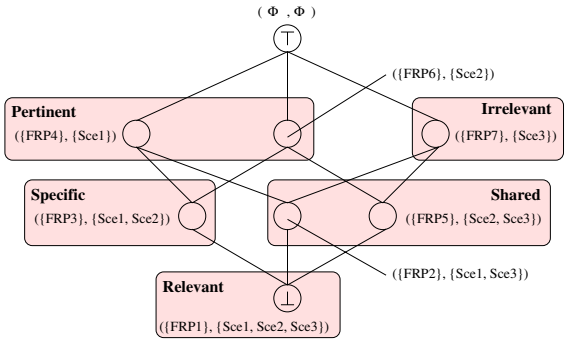


Figure 4: Categorizing the concept lattice.

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 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 FRP_3 (customize toolbox).

We notice that FRP_1 is also used in all other scenarios in the running example, so that one cannot consider FRP_1 a specific functional unit for any of Q_1, Q_2, or Q_3. FRP_3, in contrast, is used only in scenarios defining Q_1. We therefore state the hypothesis that FRP_3 is specific to Q_1 whereas FRP_1 is not. It is worth bearing in mind that this is just a hypothesis because other quality requirements might be involved to which FRP_3 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, FRP_3 is involved in both Q_2 (in Sce_2) and Q_3 (in Sce_1); then, it appears in both scenarios but nevertheless is not specific to Q_1. However, chances are high that FRP_3 is specific to Q_1 because FRP_3 is not involved when Q_2 and Q_3 are jointly expressed in Sce_3, which suggests that FRP_3 at least comes into play only when Q_1 interacts with Q_2 or Q_3. 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 Q_1 are such FRPs that are included in scenarios specifying Q_1 among other quality requirements. In the running example, both Sce_1 and Sce_2 define Q_1. FRPs in extents of concepts which contain Sce_1 or Sce_2 are therefore potentially relevant to Q_1. In our case, FRP_2, FRP_4, FRP_5, and FRP_6 are potentially relevant in addition to FRP_1 and FRP_3. FRP_7 is included only in Sce_3, which does not define Q_1. Based on the above analysis, we can identify five categories for FRPs according to their relations to Q_1. This categorization is displayed in Fig. 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 Q_1 (supporting usability) are:

- Specific: FRP_3 (customize toolbox) is specific to Q_1 because it is included in all the scenarios defining Q_1 but not in others.
- Relevant: FRP_1 (navigate shop) is relevant to Q_1 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.

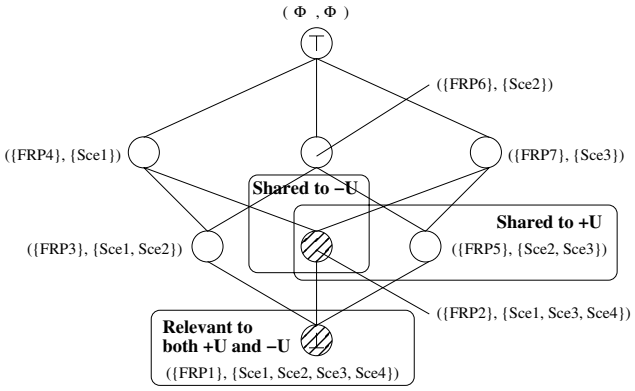


Figure 5: Detecting interferences.

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 [41]. 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 [41], 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, Sce_4, to our analysis. Sce_4 includes FRP_1 and FRP_2 (cf. Table 1), and specifies Q_4: −U (hurting usability), according to the novice user. The updated concept lattice [3] is shown in Fig. 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 [28], 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.

Implications: A key benefit of aspect-oriented RE is to detect conflicting concerns early when trade-offs can be resolved more economically [3]. 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 [41]. 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.

[3] We address the issue of reactively updating the concept lattice in Section 4.

4. 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 3 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 [22].

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 [3]. 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 3 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 3, 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 [4]. 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 ($\mathcal{O}$) to be the FRPs and the attributes ($\mathcal{A}$) to be the architectural drivers. The binary relation ($\mathcal{I}$) refers to the contribution relation [27]. Of course, we can (and should) always develop scenarios to help clarify requirements constructs and relations in the context, as discussed in Section 3.

The new instantiation ($\mathcal{O}$ = FRPs, $\mathcal{A}$ = 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.

– 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 ($\mathcal{A}$), 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.

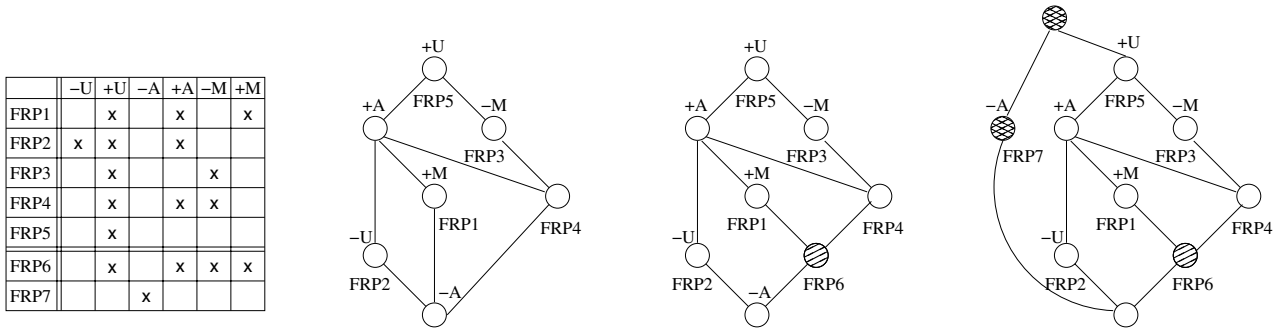
Solution: Our solution utilizes Godin et al.'s incremental lattice update algorithm, which takes as input the current lattice $\mathcal{L}$ and the new object with its attributes and outputs a new lattice $\mathcal{L}'$ that incorporates the changes [16]. The efficiency of the algorithm comes from the observation that once an initial concept lattice $\mathcal{L}$ has been constructed from the relation table $\mathcal{R}$, there is no need to maintain $\mathcal{R}$. 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, $\perp$, to the node) may now be raised in $\mathcal{L}'$ 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 [16].

Let $|\mathcal{O}|$ denote the cardinality of the set of objects $\mathcal{O}$, and $|\mathcal{A}|$ denote the cardinality of the set of attributes $\mathcal{A}$. 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 $(|\mathcal{O}| \times |\mathcal{A}|)$ [16]. 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 $|\mathcal{O}|$ [16]. Figure 6 illustrates the algorithm via the media shop example. The context is given in Fig. 6a. The attributes are a set of architectural drivers. For simplicity, we choose the quality attributes appeared in the scenarios (cf. Section 2.4), 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 1. The concept lattice for baseline FRPs is shown in Fig. 6b. The bottom ($\perp$) has intent {−A} because, through FRP_1 to FRP_5, no object relates to −A. The top ($\top$) has intent {+U} because it relates to all the FRPs in the context. The lattice after adding FRP_6 is shown in Fig. 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 Fig. 6b is preserved in this modification.

Similarly, Fig. 6d shows the concept lattice after adding FRP_7. The basic structure of Fig. 6c remains intact and incremental changes are shaded in crossed stripes. FRP_7

[4]A set of quality requirements that guide the SPL's architecture design (cf. Section 2.1). In practice, the number of architectural drivers is often less than a dozen [34].



(a) Formal context (b) Concept lattice (FRP1 through FRP5) (c) Concept lattice (FRP6 added) (d) Concept lattice (FRP7 added)

Figure 6: Updating concept hierarchy incrementally.

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 [16].

Implications: Incorporating incremental changes is essential in reactive SPL development [22]. 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 underspecified 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 Fig. 6b, for example, +M and −M are kept disjoint, showing a desired low coupling between the poles of maintainability. Adding FRP_6 (generate report) introduces an interference node, as depicted in stripes in Fig. 6c. Although this seems to destroy the modularity of the base, a semantic case analysis [29] 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.

– Issue 4: Analyze Change Impact.

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 [44]. Our approach relies on the subconcept-superconcept relation exposed in the sliced

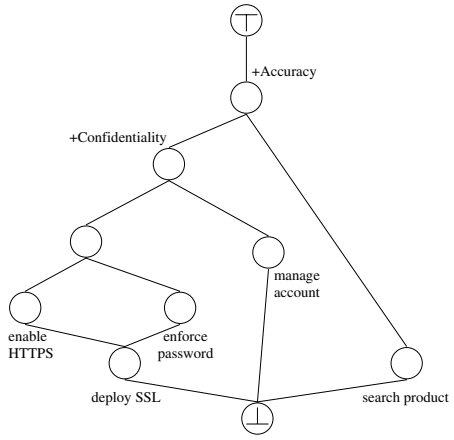


Figure 7: Analyzing trade-offs on a sliced lattice.

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 7 shows a sliced concept lattice, adapted from [27]. 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 Fig. 7,

(a) Genuine Soccer Trial (b) Genuine Soccer Pro

Figure 8: Products in the Genuine Soccer SPL.

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. CASE STUDY

We used an exploratory case study [48] 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.

5.1 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.

Table 3: Extractive Analysis Results

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

* Precision
* Relevant, Pertinent

We studied two variants in the Genuine Soccer family: the Trial version and the Pro version. The screenshot of each version is shown in Fig. 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 [29] 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 [6] to implement our concept analysis methods. The results were discussed in a half-day joint application development (JAD) workshop in FC.

5.2 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 [10]. 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 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 3.

To evaluate the effectiveness of the concern-locating results (Issue 1, cf. Section 3), 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 3: 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 3. Meanwhile, we asked the experts to manually identify the quality-specific units among the 17 FRPs. This allows us to compute precision [39] 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 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 [39], 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 [22] 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 3). We presented the concept lattice to FC's experts, and discussed 9 interferences and their causes and effects (right column of Table 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 4). 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 [48]. 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.

5.3 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 be leveraged from off-the-shelf toolkits (e.g., ToscanaJ [6]), 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 [22]. 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 [29] 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.

6. RELATED WORK

Our work 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 [3]. 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 [31].

The symmetric view of aspects was formulated in [42], but most contemporary SPL methods still exhibit "tyranny of the dominant decomposition" [42]: 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 [12]. The authors introduced some aspect-oriented patterns for implementing variable features on top of an object-oriented (OO) approach. Similarly, Liu et al. [25] 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 [32]. In [46], the authors used a traceability matrix to identify 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., [40, 1]), since legacy requirements are mostly documented in natural language. Semantic analysis was also performed to model SPL variabilities [29] and to intentionally compose the aspectual requirements [11]. In terms of reactive development, Baldwin and Coady discussed their experience of introducing a distribution implementation as aspects into the JVM [2]. 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 [15] has traditionally been applied in the field of software engineering to support software maintenance activities [43], such as program understanding [44], reengineering configuration structures [41], and code-level aspect mining [45]. 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 [38], 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 [27]. 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.

7. CONCLUSIONS

In this paper, 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" [42]. 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. And finally, we want to incorporate the current concept analysis framework into our previous SPL RE work on extracting and modeling [29], clustering [30], and managing terminological interferences [28]. The main thrust of our continuing work is to promote a set of low-threshold, lightweight techniques as a critical enabler for practitioners to capitalize on the order-of-magnitude improvements offered by SPL engineering.

8. ACKNOWLEDGMENTS

We would like to thank all the management and staff at the partner company for supporting us during the case study, for sharing not only their data but also time and expertise, and especially to Jia Wang for setting up and coordinating the meetings. We also thank John Mylopoulos and Krzysztof Czarnecki for careful comments on earlier drafts of this paper. Funding was provided by NSERC.

9. REFERENCES

[1] V. Alves, C. Schwanninger, L. Barbosa, A. Rashid, P. Sawyer, P. Rayson, C. Pohl, and A. Rummler. An exploratory study of information retrieval techniques in domain analysis. In *Int'l SPL Conf*, pages 67–76, Limerick, Ireland, September 2008.

[2] J. Baldwin and Y. Coady. Are patches cutting it? structuring distribution within a JVM using aspects. In *IBM CAS Conf*, pages 29–39, Toronto, Canada, 2005.

[3] E. Baniassad, P. C. Clements, J. Araújo, A. Moreira, A. Rashid, and B. Tekinerdoğan. Discovering early aspects. *IEEE Software*, 23(1):61–70, January/February 2006.

[4] T. Bardo, D. Elliott, T. Krysak, M. Morgan, R. Shuey, and W. Tracz. Core: A product line success story. http://www.stsc.hill.af.mil/crosstalk/1996/03/Core.asp.

[5] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addison-Wesley, 2003.

[6] P. Becker and J. Correia. The ToscanaJ suite for implementing conceptual information systems. *LNCS*, 3626:324–348, 2005.

[7] G. Birkhoff. *Lattice Theory*. Providence, RI.: Am. Math. Soc., 1940.

[8] J. D. Blaine and J. Huang. Software quality requirements: how to balance competing priorities. *IEEE Software*, 25(2):22–24, March/April 2008.

[9] J. Bosch. *Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach*. Addison-Wesley, 2000.

[10] D. Callele, E. Neufeld, and K. Schneider. Emotional requirements in video games. In *Int'l Reqs Eng Conf*, pages 299–302, Minneapolis, USA, September 2006.

[11] R. Chitchyan, A. Rashid, P. Rayson, and R. Waters. Semantics-based composition for aspect-oriented requirements engineering. In *Int'l Conf on AOSD*, pages 36–48, Vancouver, Canada, March 2007.

[12] H. Cho, L. Kwanwoo, and K. C. Kang. Feature relation and dependency management: an aspect-oriented approach. In *Int'l SPL Conf*, pages 3–11, Limerick, Ireland, September 2008.

[13] P. Clements and L. Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2001.

[14] S. R. Faulk. Product-line requirements specification (PRS): an approach and case study. In *Int'l Symp on Reqs Eng*, pages 48–55, Toronto, Canada, August 2001.

[15] B. Ganter and R. Wille. *Formal Concept Analysis: Mathematical Foundations*. Springer, 1996.

[16] R. Godin, R. Missaoui, and H. Alaoui. Incremental concept formation algorithms based on Galois (concept) lattices. *Computational Intelligence*, 11(2):246–267, 1995.

[17] IEEE Standards Board. IEEE recommended practice for software requirements specifications. 1998.

[18] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-oriented domain analysis (FODA) feasibility study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, November 1990.

[19] K. C. Kang, S. Kim, J. Lee, K. Kim, E. Shin, and M. Huh. FORM: a feature-oriented reuse method with domain-specific reference architectures. *Annals of Softw Eng*, 5:143–168, January 1998.

[20] R. Kazman, G. Abowd, L. Bass, and P. Clements. Scenario-based analysis of software architecture. *IEEE Software*, 13(6):47–55, November 1996.

[21] G. Kiczales, J. Lamping, A. Menhdhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-oriented programming. *LNCS*, 1241:220–242, 1997.

[22] C. W. Krueger. Easing the transition to software mass customization. In *Int'l Wkshp on Product-Family Eng*, pages 282–293, Bilbao, Spain, October 2001.

[23] J. Kuusela and J. Savolainen. Requirements engineering for product families. In *Int'l Conf on Softw Eng*, pages 61–69, Limerick, Ireland, June 2000.

[24] S. Liaskos, A. Lapouchnian, Y. Yu, E. Yu, and J. Mylopoulos. On goal-based variability acquisition and analysis. In *Int'l Reqs Eng Conf*, pages 76–85, Minneapolis, USA, September 2006.

[25] J. Liu, R. R. Lutz, and H. Rajan. The role of aspects in modelling product line variabilities. In *Wkshp on Aspect-Oriented Product Line Eng*, 2006.

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

[27] N. Niu and S. Easterbrook. Analysis of early aspects in requirements goal models: a concept-driven approach. *Trans. AOSD*, III:40–72, 2007.

[28] N. Niu and S. Easterbrook. So, you think you know others' goals? A repertory grid study. *IEEE Software*, 24(2):53–61, March/April 2007.

[29] N. Niu and S. Easterbrook. Extracting and modeling product line functional requirements. In *Int'l Reqs Eng Conf*, pages 155–164, Barcelona, Spain, September 2008.

[30] N. Niu and S. Easterbrook. On-demand cluster analysis for product line functional requirements. In *Int'l SPL Conf*, pages 87–96, Limerick, Ireland, September 2008.

[31] N. Niu, S. Easterbrook, and Y. Yu. A taxonomy of asymmetric requirements aspects. *LNCS*, 4765:1–18, 2007.

[32] N. Niu, Y. Yu, B. González-Baixauli, N. Ernst, J. Leite, and J. Mylopoulos. Aspects across software life cycle: a goal-driven approach. *Trans. AOSD*, (to appear), 2008.

[33] B. Nuseibeh and S. Easterbrook. Requirements engineering: a roadmap. In *Conf on Future Softw Eng*, pages 35–46, Limerick, Ireland, June 2000.

[34] I. Ozkaya, L. Bass, R. L. Nord, and R. S. Sangwan. Making practical use of quality attribute information. *IEEE Software*, 25(2):25–33, March/April 2008.

[35] K. Pohl, G. Böckle, and F. van der Linden. *Software Product Line Engineering: Foundations, Principles, and Techniques*. Springer, 2005.

[36] R. Prieto-Díaz. Domain analysis: an introduction. *Softw Eng Notes*, 15(2):47–54, April 1990.

[37] A. Rashid, A. Moreira, and J. Araújo. Modularisation and composition of aspectual requirements. In *Int'l Conf on AOSD*, pages 11–20, Boston, USA, March 2003.

[38] D. Richards. Merging individual conceptual models of requirements. *Reqs Eng Journal*, 8(4):195–205, 2003.

[39] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, 1983.

[40] A. Sampaio, A. Rashid, R. Chitchyan, and P. Rayson. EA-Miner: towards automation in aspect-oriented requirements engineering. *Trans. AOSD*, III:4–39, 2007.

[41] G. Snelting. Reengineering of configurations based on mathematical concept analysis. *ACM Trans. Softw Eng and Methodology*, 5(2):146–189, April 1996.

[42] P. Tarr, H. Ossher, W. Harrison, and S. M. Sutton. N degree of separation: multi-dimensional separation of concerns. In *Int'l Conf on Softw Eng*, pages 107–119, Los Angeles, USA, May 1999.

[43] T. Tilley, R. Cole, P. Becker, and P. Eklund. A survey of formal concept analysis support for software engineering activities. *LNCS*, 3626:250–271, 2005.

[44] P. Tonella. Using a concept lattice of decomposition slices for program understanding and impact analysis. *IEEE Trans. Softw Eng*, 29(6):495–509, June 2003.

[45] P. Tonella and M. Ceccato. Aspect mining through the formal concept analysis of execution traces. In *Working Conf on Reverse Eng*, pages 112–121, Delft, The Netherlands, November 2004.

[46] K. van den Berg, J. M. Conejero, and J. Hernández. Analysis of crosscutting in early software development phases based on traceability. *Trans. AOSD*, III:73–104, 2007.

[47] D. M. Weiss and C. T. R. Lai. *Product-Line Engineering: A Family-Based Software Development Process*. Addison-Wesley, 1999.

[48] R. Yin. *Case Study Research: Design and Methods*. Sage Publications, 2003.