Analyzing Evolution of Variability in a Software Product Line: from Contexts and Requirements to Features

Xin Peng[1], Yijun Yu[2], Wenyun Zhao[1]
[1] School of Computer Science, Fudan University, China
[2] Department of Computing, The Open University, UK
{pengxin, wyzhao}@fudan.edu.cn, {y.yu}@open.ac.uk

Abstract

In the long run, features of a software product line (SPL) evolve with respect to changes in stakeholder requirements and system contexts. Neither domain engineering nor requirements engineering handles such co-evolution of requirements and contexts explicitly, making it especially hard to reason about the impact of co-changes in complex scenarios. In this paper, we propose a problem-oriented and value-based analysis method for variability evolution analysis. The method takes into account both kinds of changes (requirements and contexts) during the life of an evolving software product line. The proposed method extends the core requirements engineering ontology with the notions to represent variability-intensive problem decomposition and evolution. On the basis of problem-orientation, the analysis method identifies candidate changes, detects influenced features, and evaluates their contributions to the value of the SPL. The process of applying the analysis method is illustrated using a concrete case study of an evolving enterprise software system, which has confirmed that tracing back to requirements and contextual changes is an effective way to understand the evolution of variability in the software product line.

1. Introduction

Software product lines (SPL) are longer term investments than individual products. Their promise is to construct high-quality products quicker with less cost [22, 17] by *proactive reuse*. Central to SPL development is the notion of *commonality* and *variability*. They are usually modeled by features, which are distinctively identifiable functional abstractions, and organized into feature models as an AND/OR graph, where AND nodes indicate mandatory features and OR nodes indicate variation features selectable for different applications [9]. In feature models, typical variation features include optional, alternative (i.e., exclusive OR) and inclusive OR features. In the long-term evolution of an SPL, variability constantly evolves with the extension of scopes and changes in the applications. Capturing, understanding and predicting variability evolution are *all* important to achieve the SPL promise.

Changes to a feature model reflect changes in the *requirements* of the stakeholders or changes in the domain *contexts* of the software products. An understanding of the relationship among these

changes may help one prepare for possible changes in the future, to control the risk of unexpected changes, and to discover the missing rationale for the evolution of variability in the past.

However, the analysis is difficult due to the lack of traceability between requirements, contexts and features. Domain engineering approaches [9, 17, 22, 20] interpret the variability across different application contexts. Volatile requirements approaches [25, 18] express the variability in changing requirements. Nonetheless, little has been done to analyze the causes of a change in the feature by looking at changes of both contexts and requirements [1].

Considering co-changes to contexts and requirements, our goal is to create a problem-oriented method for analyzing variability evolution of an SPL. To this end, we first establish the intentional and contextual bases for feature variations and variability evolution using the core requirements engineering ontology [13]. Then we propose a value-based analysis method for both evolution history analysis and future evolution prediction, following the perspective of value-based software engineering [4] that considers economic aspects (such as business value) within the duration of the whole software development. We categorize typical variability co-evolution patterns in terms of context and requirement changes into a few canonical rules, which are used by our impact analysis algorithm to identify the impacted features. To demonstrate the effectiveness of the overall approach, we conduct a case study on the computer-aided grading system (CAGS-PL), an enterprise product line for different kinds of official and intra-school examinations widely used in China.

The main contribution of our work is threefold. First, we propose to analyze variability evolution from the prospect of context and requirement changes, and present a set of evolution rules that relate changes of contexts and requirements to those of feature variations. Second, we develop an algorithm to analyze the impact scope to classify contexts or requirements change in the variability evolution. Third, we illustrate a systematic process using these rules and the impact analysis algorithm to help identify possible changes in contexts and requirements and rank their influences on features.

The remainder of the paper is organized as follows. Section 2 introduces the background of the core ontology in requirements engineering and gives an account of the brief evolution history of CAGS-PL. Section 3 provides a formally defined theory of variability evolution as an extension to the core requirements engineering ontology. Based on the variability evolution theory, in Section 4, we propose our value-based analysis method for variability evolution, which is supported by a change impact analysis algorithm. Section 5 details the case study and Section 6 evaluates the benefits of our problem-oriented analysis method and presents a discussion on the completeness and correctness of our method. Section 7 presents the work in relation to the requirement evolution and the variability analysis literature. Finally, Section 8 concludes the paper.

2. Background

In this section, we start with introducing the background of the core ontology in requirements engineering, which forms the basis of our approach. We then introduce the CAGS-PL case study and its evolution history, which will later be used as a running example to illustrate our approach.

Figure 1. Graphical syntax of a problem frame

2.1. The core ontology in RE

The Problem Frames (PF) approach [12] classifies software development problems into an analytical structure that relates requirements to the contexts surrounding the system. A specification of a requirement is interpreted as transforming the context domains in a physical world by the system-to-be (also called interchangeably as a *solution* or a *machine*). Zave and Jackson [30] define requirement engineering as the problem of finding a way to satisfy the following entailment:

$$\frac{?}{W, S \vdash R} \langle\langle problem \rangle\rangle \tag{1}$$

where requirements R are statements that confine the optative properties in the system S and the indicative properties of the contexts W in which the system operates. Thus, the requirements problem amounts to finding a solution S that satisfies the given requirements R for given domain assumptions W [30]. The semantics of requirements engineering problems are further presented as Problem Frames semantics in problem-oriented software engineering [10].

The structure and relationships among the problem domains are typically presented by a problem diagram in the PF approach. A generalized problem diagram [12] shown in Figure 1 includes a requirement, a problem world, and a solution to be developed. These problem elements are all to be understood in terms of physical phenomena rather than purely mathematical abstractions [12]. The requirement is a predicate or a condition on the problem world that the solution is required to bring about. It is shown by a dashed oval, indicating its intangible nature. The problem world consists of a set of domains that are tangible, thus are represented by solid rectangles. A rectangle with a single vertical stripe denotes a "designed" domain, which is to be obtained as a solution to a subproblem. A rectangle without any stripe denotes a given domain whose existence is independent of the solution. A solution (or a machine domain) is shown as a rectangle with two vertical stripes, linking with the problem world at interfaces of shared phenomena (typically events and states). The dashed link between requirements and the problem world domain denotes either a reference or a constraint by the requirement on the physical phenomena of the problem world.

In parallel, Dardenne et al [8] initiates the KAOS approach to refine requirements into logical rules relating to stakeholder goals. A goal of the system can be AND- or OR-decomposed into one to more subgoals, such that one can assess the satisfaction of one goal by checking the satisfaction of subgoals. Mylopoulos et al [19] further introduces the concept of softgoals, the goals that does not have a clearcut criteria for satisfaction, to represent quality requirements as nonfunctional requirements (NFR), whilst regarding functional requirements as hard goals.

Although problem frames and goal models above are complementary, none of the them alone can capture a complete model of the requirements of stakeholders. In order to allow for the partial fulfilment of nonfunctional requirements, recently Jureta et al [13] propose a more unified

representations and semantics for requirements, also known as the core ontology of requirements engineering, by extending the requirements part of rule (1) as follows:

$$\frac{W, S \vdash F, Q}{W, S \vdash R} \langle\langle refinement \rangle\rangle \tag{2}$$

where F denotes the functional requirements, Q denotes the quality requirements. In the very definition of the core ontology, the semantics of Q can be richer than quality requirements, e.g., to include the emotional and psychological modes [13]. However, it is our belief that handling quality requirements alone is the necessary step in this work.

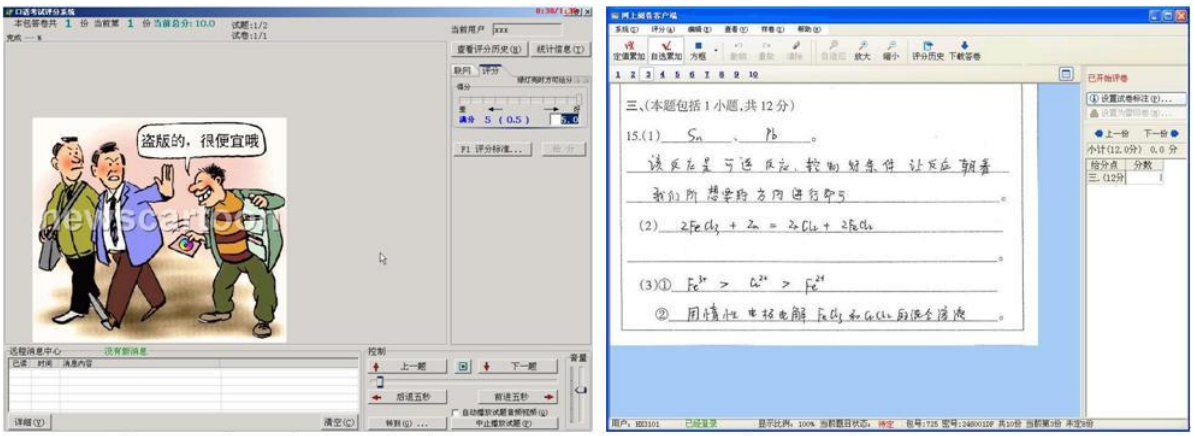
Goal models represent the intentions which may or may not be fulfilled, while feature models represent the variability as the system-to-be [28]. Therefore, one can use goal models to capture the functional and quality requirements, and use feature models to structure the solutions to be developed. Using feature models to capture the variability in the solution domains S, in this paper, we consider various changes that might happen to the elements in the core ontology and propose an extension that is necessary to represent and detect the variability evolution in S during the long-term SPL evolution with respect to changes in W and Q, while maintaining the top-level functional requirements F intact.

2.2. The CAGS product line

We use a software product line system to illustrate the concepts of co-changes analysis. The computer-aided grading system (CAGS) is a commercial system for oral or written examinations. During the exams, the system collects and preprocesses the answers into electronic answer sheets before they are to be marked manually by the teachers. Although it was developed by a small team, this product line supports a wide range of applications, including both oral and written examinations. The two different screenshots in Figure 2 illustrate its use for written examinations. Some products support intra-school examinations and some support formal official examinations, some support regular school students and some other for vocational skill trainees.

Common to the products in the CAGS-PL, the measurable quality requirements are comfortability, standardization and total cost of ownership (TCO). Comfortability is provided by anything that can make users feel comfortable, which is wider than user-friendliness. Key to comfortability in this domain are error-freedom and reliability. Error-freedom aims to have no grading errors caused by personal or system mistakes, whilst reliability aims to have the smallest mean time of continuous service delivery without failures. They can be achieved by employing barcode-based solutions which can greatly reduce the mistakes in examinee identification. Standardization aims to make the examination conforming to standards, for instance, A3-sized answer sheets must be used in official written examinations in China. TCO includes both the direct and indirect costs on the customer side for operating and maintaining the system, for instance, the cost of scanner is the major part of TCO in application for written examinations.

Initially developed for spoken English examinations, the CAGS was a by-product of the oral testing system for teachers to grade audio answers produced in the oral exams. During the early phase, majority of customers were Chinese universities who adopted a computer-aided oral testing. Since its inception in 2003 (phase 1), the CAGS has evolved into a new product line CAGS-PL and has gone through three more phases of maintenance as its users base and the scope



(a) CAGS for spoken examination (b) CAGS for written examination

Figure 2. Screenshots of CAGS application products

of application domain gets wider. In 2004 (phase 2), the system was adopted in some official examinations for college entrance tests, in addition to the original oral testing. In late 2006 (phase 3), the CAGS-PL products for written examination were released, targeting at regional-level formal examinations within cities for senior high-school entrance examinations. Later in 2008 (phase 4), the CAGS products for junior middle schools were released for their intra-school regular written examinations, including monthly quizs and seminal exams.

3. Rationale of variability evolution

In this section, we first give an informal introduction to the notion of tracing feature variability to requirements and contexts, then describe the canonical form of the problem derivation for contexts or requirements changes, classifying the rationale behind the evolution of feature variability.

3.1. Tracing feature variability to requirements and contexts

Analyzing feature variations according to their intentional or contextual factors may help one understand the origin of variability in features, since both intentional and contextual variability can arise in requirements engineering [15, 14].

Intentional variability represents the diversities of requirements among different products in a software product line. Different products usually share similar basic functions as mandatory features, but they may have different choices of variant features due to the tradeoff decisions to meet quality or nonfunctional requirements. In order to accommodate different preferences of stakeholders, these tradeoffs must be kept in the product line, resulting in quality-related variability [20]. For example, in CAGS-PL, all products share the same basic functions such as answering preparation, dispatching, grading, querying and calculating statistics. These functions all have different preferences in terms of quality requirements. In some applications, TCO and comfortability are preferred, whilst in others error-freedom and standardization are preferable to TCO and comfortability. These quality preferences usually lead to variations in corresponding

features, e.g. both barcode-based and blackened-code-based examinee recognition are available as alternative features for examinee identification. The former has a much better precision at a higher cost: about 0.007 US dollars per barcode.

Contextual variability represents the diversities of entities or contextual conditions in the problem world among different applications. They may request different solutions for the same requirements. We identify two kinds of contexts that affect the solution for the specified requirements: (i) The *driving* context determines whether a solution makes sense at all. For example, CAGS uses the Answer Sheets in the products for written exams and uses Audio Answers in the products for oral exams. They require different solutions for preparation and display of the answers, although answer scanning and other image-related treatment are required for the Answer Sheet alternative, they become meaningless to the audio-based preparation of answers. (ii) Complementary to the driving contexts, the *supporting* context maintains the solution for given functional requirements whilst either strengthening or weakening certain quality requirements. For example, the closed and trusted network is a stronger supporting context for the security requirement than the open network. Therefore, a supporting context can also affect the variations in an SPL. In Section 3.3 we will show that supporting context is often connected to quality preferences in the variability analysis.

From a static viewpoint, feature variations can be traced back to the differences in requirements and contexts. From a dynamic viewpoint, variability evolution can be interpreted by requirement and context changes. It is our belief that classifying the different relationships between variations in specifications of the features and diversities in quality requirements and contexts, one can trace their co-changes in the evolution. This enables a further analysis of the impact of changes to better understand and plan variability evolution. In the remainder of the paper, we show how such a combined view helps in analyzing the evolution of variability.

3.2. Basic definitions

To facilitate the proposed variability analysis, we refine the requirements semantics and representation of the Rule (2) with quality preferences and define the requirements problem in our method formally as follows.

Definition 1. *A requirements problem p is a 6-tuple* $< W_d, W_s, S, F, Q^+, Q^- >$ *representing that solution S with domain assumptions given by W satisfies the functional requirements F and both positive and negative preferences on nonfunctional qualities Q. More specifically,* W_d *is the set of relevant driving contexts that are characterised in Definition 3,* W_s *are relevant supporting contexts characterised in Definition 4, and* $W_h = W \setminus (W_d \cup W_s)$ *are irrelevant or hidden domain assumptions;* Q^+ *denotes the quality attributes that are positively influenced (helped),* Q^- *denotes the quality attributes that are negatively influenced (hurt), and* $Q^0 = Q \setminus (Q^+ \cup Q^-)$ *are indifferent ones. For the analysts, the hidden domain assumptions* W_h *and indifferent quality requirements* Q^0 *can be ignored. Therefore, following the problem representation in rule (1) and rule (2), the problem can also be represented by the following formula:*

$$W_d, W_s, S \vdash F, Q^+, Q^- \tag{3}$$

where W_d*,* W_s*,* F*,* Q^+ *and* Q^- *are represented by a proposition in the conjunctive normal form (CNF) and S is represented by a functional feature.*

In this definition of a problem, Q^+ and Q^- are divided from the general quality requirements Q in Rule (2) to represent the quality tradeoff involved in requirements refinement. The specific quality requirements Q^+ (resp. Q^-) denotes a satisfaction degree that is higher (resp. lower) than the that of the average solutions to F. Intuitively, Q^+ and Q^- can be understood as the quality requirements preferred and conceded in the solution respectively. For example, "Q^+: PS(standardization)" and "Q^-: PD(TCO)" involved in Formula (4) represent the preference of "standardization over TCO" in the fulfillment of AnswerScan, with PS and PD respectively meaning 'Partially satisfied' and 'Partially Denied'. An implication of the problem definition is that F should be meaningful and can be interpreted in the given contexts. For example, functional requirement AnswerScan is meaningful only with the context setting AnswerSheet (AnswerScan is meaningless in an oral examination application with corresponding context setting of AudioAnswer).

It should be emphasized that "$\vdash$" in our problem representation does not mean logical implication. It denotes a requirements satisfaction relation between a solution with certain context assumptions and a requirements statement with both the functional and nonfunctional parts. Accordingly, "$\nvdash$" means the requirements on the right are meaningless or cannot be satisfied by the solution and context assumptions on the left. According to the tuple-based Definition 1, all of the elements W_d, W_s, S, F, Q^+ and Q^- in a problem formula are specific, as they denote different aspects of a problem. And the comma used in the formula denotes the separator for different problem elements. Therefore, "$\vdash$" is a domain-independent relation and is not a partial order.

Then one can use logical implication between context or requirement propositions to help a formal variation analysis. In order to distinguish from the requirement satisfaction relation "$\vdash$", we use "$\mapsto$" to denote the logical implication between context or requirement propositions. Context propositions usually involve predicates about the existence or the properties of certain entities. For example, the following implication tells that the context specified on the left hand side is a specialization of the right hand side: (A3 answer sheet)$\wedge$ (networked grading)$\mapsto$(audio answer $\vee$ answer sheet) $\wedge$ (networked grading). A quality requirement has a multi-valued label to indicate the degree of its satisfaction [29]. To reason about quality requirements, we introduce the satisfaction level predicates of a quality requirement Q, such as FullySatisfied (FS), PartiallySatisfied (PS), PartiallyDenied (PD) and FullyDenied (FD) etc., in quality requirement representation. To reason about quality requirements, logical implications are considered as comparisons between discretised quality levels, e.g. "FS(TCO: 1,500)$\mapsto$FD(TCO: 12,000)" holds.

Based on Definition 1, we further define the concept of stronger/weaker solutions.

Definition 2. *Given functional and quality requirements F, Q and the contexts W, a solution S_1 is* stronger than *another solution S_2 if and only if*

$$\frac{W, S_2 \vdash F', Q'^+, Q'^- \quad W, S_2 \nvdash F, Q^+, Q^-}{W, S_1 \vdash F, Q^+, Q^-} \langle\langle stronger \rangle\rangle$$

as long as the followings are satisfied:

1. *F are* stronger than *F', i.e.:*

$$F \mapsto F' \wedge F' \not\mapsto F$$

2. $F = F'$ *and* Q^+/Q^- *are* better than Q'^+/Q'^-, *i.e.:*

$$(Q^+ \wedge Q^- \mapsto Q'^+ \wedge Q'^-) \wedge (Q'^+ \wedge Q'^- \not\mapsto Q^+ \wedge Q^-)$$

Since functional requirements are the basis for a solution, we select a subset of *driving contexts* to characterizes the causes of changes to the functional requirements.

Definition 3. *Contexts* W_d *are the* driving *contexts for a given solution and functional requirements* F*, iff their changes* W'_d *can violate the satisfaction of functional requirements* F*:*

$$\frac{W'_d, (W \setminus W_d), S \not\vdash F}{W_d, (W \setminus W_d), S \vdash F} \langle\langle driving \rangle\rangle$$

Here W'_d *denotes a different driving context from* W_d *and* $W \setminus W_d$ *means all the other contexts in* W *except* W_d*.*

To capture the contexts whose changes still support the functional requirements while affecting the satisfaction of quality requirements, we denote them as *supporting* contexts W_s.

Definition 4. *Contexts* W_s *are the* supporting *contexts for a given solution* S*, given functional requirements* F *and quality requirements* Q*, iff their changes* W'_s *can affect the satisfaction of quality requirements* Q^+ *or* Q^-*:*

$$\frac{W_d, W'_s, S \vdash F, Q'^+, Q'^-}{W_d, W_s, S \vdash F, Q^+, Q^-} \langle\langle supporting \rangle\rangle$$

where $(Q^+ \wedge Q^- \not\mapsto Q'^+ \wedge Q'^-) \vee (Q'^+ \wedge Q'^- \not\mapsto Q^+ \wedge Q^-)$ *holds.*

The example in Formula (4) presents the problem representation using the canonical form given in Rule (3). In the example formula, the prefix before each colon denotes the corresponding element in the problem formula. It states that given the driving context AnswerSheet, the solution SinglePageScan is provided with supporting contexts A3-Scanner to implement the requirement AnswerScan. The influenced quality requirements Q^+ and Q^- show that the solution helps the quality requirement standardization but hurts TCO. In this problem, if the driving context AnswerSheet is not satisfied, e.g. in an oral examination application, the requirement AnswerScan will be meaningless. The supporting context A3-Scanner is required in the solution, and further leads to the quality preference of standardization over TCO: although A3-sized answer sheet is standard, an A3-scanner is much more expensive than an A4-scanner.

$$\begin{array}{l} W_d : AnswerSheet, W_s : A3Scanner, S : SinglePageScan \vdash \\ F : AnswerScan, Q^+ : PS(standardization), Q^- : PD(TCO) \end{array} \tag{4}$$

3.3. Variability-intensive problem decompositions

For a product line, the features including those variations already identified are typically specified on the basis of the current assumptions on the requirements and contexts in the domain. In order to capture the requirement and context basis for further evolution analysis, we propose a

set of problem decomposition rules to establish a problem decomposition structure for the whole domain. We assume that all the products share the same initial contexts and requirements, including a functional requirement and some quality requirements. Starting from a common initial problem, a series of variability-intensive problem decompositions can be derived using those rules. Here, variability-intensive problem decomposition is to refine a general problem into more specific sub-problems by certain requirement/context specializations. In problem refinements, feature variations are identified as the solutions for different sub-problems of the OR-decompositions, and related assumptions on the requirements and contexts can be captured as the basis of variability evolution analysis.

Given a problem described as Rules (1) and (2), we use the problem transformation semantics similar to that of Hall et al [11] as follows:

$$\frac{\bigwedge_i (W_i, S_i \vdash R_i) \wedge (W \mapsto \bigwedge_i W_i) \wedge (\bigwedge_i R_i \mapsto R)}{W, S \vdash R} \langle\langle AND \rangle\rangle \tag{5}$$

$$\frac{\bigwedge_i (W_i, S_i \vdash F, Q_i) \wedge \bigwedge_i (W_i \mapsto W) \wedge \bigwedge_i (Q_i \mapsto Q)}{W, S \vdash F, Q} \langle\langle OR \rangle\rangle \tag{6}$$

The AND-decomposition rule (5) denotes that S is the solution only when all S_i, for $i = 1, \ldots, n$ are specified in *compatible* contexts and requirements. Similarly, the OR-decomposition rule (6) denotes that any variant solution S_i can be a solution to S in specific requirement/context settings, and the conjunctive connection of variant problems indicates that a complete refinement to a general problem should cover all the specialized sub-problems under different requirement and context settings.

Now consider different kinds of requirement and context specializations in our canonical rule (3), we can identify two kinds of variability-intensive problem decompositions as follows:

$$\frac{\bigwedge_i (W_{di}, W_s, S_i \vdash F, Q^+, Q^-)}{W_d, W_s, S \vdash F, Q^+, Q^-} \langle\langle DrivingOR \rangle\rangle \tag{7}$$

In rule (7), $W_{d1}, \ldots, W_{dn}$ are disjoint specializations of W_d (i.e., $W_{di} \mapsto W_d \wedge \forall i, j : W_{di} \neq W_{dj}$), and they demand different solutions to satisfy the same requirement. Formula (8) specifies an example of OR-decomposition by driving context using rule (7). It tells that the two specializations of the driving context Answer, AudioRecord and AnswerSheet, require different solutions to implement the same functional requirement AnswerPreprocess.

$$\frac{\begin{array}{l} W_d : AudioRecord, W_s :, S : AudioAnsPrepare \vdash F : AnswerPreprocess, Q^+ :, Q^- : \bigwedge \\ W_d : AnswerSheet, W_s : Scanner, S : ImgAnsPrepare \vdash F : AnswerPreprocess, Q^+ :, Q^- : \end{array}}{W_d : Answer, W_s :, S : AnswerPrepare \vdash F : AnswerPreprocess, Q^+ :, Q^- :} \tag{8}$$

The next rule concerns OR-decompositions for different tradeoffs of quality requirements.

$$\frac{\begin{array}{l}\bigwedge_i(W_d, W_{si}, S_i \vdash F, Q_i^+, Q_i^-)\wedge \\ \forall j \neq i : ((Q_i^+ \wedge Q_i^- \not\Rightarrow Q_j^+ \wedge Q_j^-) \wedge (Q_j^+ \wedge Q_j^- \not\Rightarrow Q_i^+ \wedge Q_i^-))\end{array}}{W_d, W_s, S \vdash F, Q^+, Q^-} \langle\langle PreferenceOR \rangle\rangle \tag{9}$$

Q_i^+/Q_i^- for $i = 1, \ldots n$ are different and exclusive quality tradeoffs for the same set of quality requirements as those of Q^+/Q^-, i.e. there is no quality tradeoff that is absolutely better than another one. These alternative solutions finally will be chosen by the customers using an application-specific utility function reflecting their quality preferences, e.g. with a weight schema for different quality requirements. In these OR-decompositions by quality preferences, corresponding supporting contexts for each sub-problem (W_{si}) may also be different.

Formula (10) specifies an example of preference-driven OR-decomposition using Rule (9). It tells that the two variant solutions of SinglePageScan and MultiPageScan are provided to implement AnswerScan with different quality preferences of "standardization over TCO" and "TCO over standardization". According to rule (9), neither of the two variant solutions is absolutely better than the other one, otherwise we can always eliminate the worst one in the previous variability analysis.

$$\frac{\begin{array}{l}W_d : AnswerSheet, W_s : A3Scanner, S : SinglePageScan \vdash F : ImgAnsScan, \\ Q^+ : PS(standardization), Q^- : PD(TCO) \bigwedge W_d : AnswerSheet, W_s : A4Scanner, \\ S : MultiPageScan \vdash F : ImgAnsScan, Q^+ : PS(TCO), Q^- : FD(standardization)\end{array}}{W_d : AnswerSheet, W_s : Scanner, S : AnswerScan \vdash F : ImgAnsScan, Q^+ :, Q^- :} \tag{10}$$

There are no OR-decompositions for different supporting contexts. Actually, supporting contexts usually appear together with quality preferences as the driving forces of specialized problem decompositions, see Rule (9). In this kind of decompositions, quality preferences are always the decisive factors. If no quality preferences are involved in the decomposition, i.e. there is one sub-problem such that all its quality requirements are better satisfied than those of others, then the stronger solution (see Definition 2) can always replace those weaker ones and variability will not emerge.

It should be noted that problem decompositions in an evolving SPL are not invariable. Existing problem decompositions on Rules (5)-(9) only reflect the current situation of the SPL. Along with SPL evolution like scope extension and technical evolution, they may be reevaluated and new variations may be introduced. For example, considering a new driving context for a problem decomposed using Rule (7) may request a new variant feature to be introduced.

3.4. Variability evolution rules

Rules (6), (7) and (9) provide only a static interpretation for feature variations. From these rules, one can observe that the current variability status of a solution is based on certain driving and supporting contexts and quality tradeoffs. Taking a dynamic perspective with changing requirements and contexts, on the other hand, one may discover the patterns and rules of variability evolution through the variability-intensive problem decompositions. Three variability evolution patterns are described as follows, each includes two problem-derivation rules.

3.4.1. Mandatory becomes Optional

In dynamic variability evolution, a mandatory feature can turn into optional, which indicates that the feature is no longer applicable to the emerging requirements or contexts. The following rules reflect different variability evolution patterns. The left and right sides of an *evolve* arrow in these rules represent before and after the change respectively. The label above the arrow indicates the factors such as the changed requirements or contexts.

Rule (11) describes the case of "Mandatory to Optional" evolution caused by a newly introduced driving context. In this case, the requirements become meaningless under the newly introduced driving context W_{d2}. For example, AnsMark, which enables a teacher to add marks on the answer images, is a mandatory feature for written answers. It will become optional after oral answers are introduced, as the original requirement is no longer meaningful for the oral answers.

$$\frac{W_{d1}, W_s, S_1 \vdash F, Q^+, Q^-}{W_d, W_s, S \vdash F, Q^+, Q^-} \xrightarrow[evolve]{W_{d2} \neq W_{d1}} \frac{W_{d1}, W_s, S_1 \vdash F, Q^+, Q^- \bigwedge \forall S_2 : W_{d2}, W_s, S_2 \nvdash F}{W_d, W_s, S \vdash F, Q^+, Q^-} \tag{11}$$

The other kind of "Mandatory to Optional" evolution is caused by newly involved quality preferences. As stated in Rule (12), a feature introduced only for quality requirements (the functional requirement F is null) will evolve to be optional if the quality tradeoff is unacceptable for the new quality preferences. For example, FastForward, introduced in the initial applications for intra-school examinations for better comfortability in oral answer display, turns out to be optional when "error-freedom over comfortability" emerges with the newly introduced applications for official examinations.

$$\frac{W_d, W_{s1}, S_1 \vdash Q_1^+, Q_1^-}{W_d, W_s, S \vdash Q^+, Q^-} \xrightarrow[evolve]{Q_2^+ \neq Q_1^+, Q_2^- \neq Q_1^-} \frac{W_d, W_{s1}, S_1 \vdash Q_1^+, Q_1^- \bigwedge W_d, W_{s1}, S_1 \nvdash Q_2^+, Q_2^-}{W_d, W_s, S \vdash Q^+, Q^-} \tag{12}$$

3.4.2. Mandatory/Optional becomes Alternative

This pattern means a mandatory or optional feature without any variants is turned into an alternative feature with two variants, i.e. the existing feature and the newly introduced variant. Similar to the "Mandatory to Optional" rules, one can identify two evolution rules for this kind of evolution when the two aspects of requirement and contextual changes are considered.

$$\frac{W_{d1}, W_s, S_1 \vdash F, Q^+, Q^-}{W_d, W_s, S \vdash F, Q^+, Q^-} \xrightarrow[evolve]{W_{d2} \neq W_{d1}} \frac{W_{d1}, W_s, S_1 \vdash F, Q^+, Q^- \bigwedge W_{d2}, W_s, S_2 \vdash F, Q^+, Q^-}{W_d, W_s, S \vdash F, Q^+, Q^-} \tag{13}$$

In Rule (13), the requirements F, Q^+, Q^- are still desired in the new driving context W_{d2}, so a new variant solution should be introduced to adapt to the new context. For example, before written answers are introduced, AudioAnsPrepare is the only solution for the requirement AnswerPrepare. After written answers are introduced, ImgAnsPrepare, which provides the preparation for written answers, is introduced and the parent feature AnswerPrepare becomes an alternative feature.

$$\frac{W_d, W_{s1}, S_1 \vdash F, Q_1^+, Q_1^-}{W_d, W_s, S \vdash F, Q^+, Q^-} \xrightarrow[evolve]{Q_2^+ \neq Q_1^+, Q_2^- \neq Q_1^-} \frac{W_d, W_{s1}, S_1 \vdash F, Q_1^+, Q_1^- \bigwedge W_d, W_{s2}, S_2 \vdash F, Q_2^+, Q_2^-}{W_d, W_s, S \vdash F, Q^+, Q^-} \tag{14}$$

In rule (14), the functional requirement F is still desired after evolution under the new quality preference Q_2^+/Q_2^-. Therefore, a new variant solution should be introduced to support the new quality tradeoff, perhaps with different supporting contexts. For example, the initial solution for AnswerScan in written answer preparation is SinglePageScan, which supports standard A3-sized answer sheets but requires expensive A3 scanners. Therefore, after the new preference of "TCO over standardization" is introduced, a new variant solution MultiPageScan is introduced, requiring much cheaper A4 scanners than those with non A4-sized answer sheets.

3.4.3. New Variant arises

As shown in Rules (15) and (16), the main difference to Rules (13) and (14) is that the problem before evolution has already been OR-decomposed or several alternative sub-features already exist. In such cases, a new variant feature can be introduced as the sub-feature of an existing alternative feature.

$$\frac{\bigwedge_{i=1}^{n}(W_{di}, W_s, S_i \vdash F, Q^+, Q^-)}{W_d, W_s, S \vdash F, Q^+, Q^-} \xrightarrow[evolve]{\bigwedge_{i=1}^{n}(W_{d(n+1)} \neq W_{di})} \frac{\bigwedge_{i=1}^{n+1}(W_{di}, W_s, S_i \vdash F, Q^+, Q^-)}{W_d, W_s, S \vdash F, Q^+, Q^-} \tag{15}$$

$$\frac{\bigwedge_{i=1}^{n}(W_d, W_{si}, S_i \vdash F, Q_i^+, Q_i^-)}{W_d, W_s, S \vdash F, Q^+, Q^-} \xrightarrow[evolve]{\bigwedge_{i=1}^{n}(Q_{n+1}^+ \neq Q_i^+), \bigwedge_{i=1}^{n}(Q_{n+1}^- \neq Q_i^-)} \frac{\bigwedge_{i=1}^{n+1}(W_d, W_{si}, S_i \vdash F, Q_i^+, Q_i^-)}{W_d, W_s, S \vdash F, Q^+, Q^-} \tag{16}$$

3.4.4. Summary

To summarize, four kinds of factors that may bring in new variations:

1. New driving context (new W_d).
2. Weakened supporting context (weakened W_s).
3. Insufficient quality satisfaction (insufficient Q^+)
4. Unacceptable quality concession (unacceptable Q^-)

These evolution factors, together with the existing problem structures, theoretically determine whether variability evolution will emerge, as catagorized in Table 1. It can be seen that supporting context is always accompanied by different quality preferences in variability evolution. And the rationales for the second and the third evolution types are similar, and the difference is whether the feature is already an alternative feature before the evolution.

4. Variability evolution analysis

Our variability evolution analysis method has two objectives:

1. to identify possible context or requirement changes that may result in feature variability evolution;
2. to support design decisions by ranking candidate changes in a value-based perspective.

Table 1. Elementary variability evolution types

Evolution Type	Rationale	Rule	New Feature
Mandatory becomes Optional	the requirements are meaningless in the new introduced driving context W_{d2}	(11)	N/A
	existing solution is only for quality requirements and the quality tradeoff is unacceptable for the new quality preference Q_2^+/Q_2^-	(12)	N/A
Mandatory/Optional becomes Alternative	existing solution for the requirements is not compatible with the new driving context W_{d2}	(13)	a new solution to achieve the same requirements under the new driving context
	W_{s1} can not be satisfied in the changed context (a weaker supporting context W_{s2} is provided)	(14)	a stronger solution to achieve the same functional requirements with a weaker context support and new quality tradeoffs
	the functional requirements are still desired but the quality tradeoff is unacceptable for the new quality preference Q_2^+/Q_2^-	(14)	a new solution for the same functional requirements with desired quality tradeoffs
New Variant arises	existing solution for the requirements is not compatible with the new driving context $W_{d(n+1)}$	(15)	a new solution to achieve the same requirements under the new driving context
	W_{s1} can not be satisfied in the changed context (a weaker supporting context $W_{s(n+1)}$ is provided)	(16)	a stronger solution to achieve the same functional requirements with a weaker context support and new quality tradeoffs
	the functional requirements are still desired but the quality tradeoff is unacceptable for the new quality preference Q_{n+1}^+/Q_{n+1}^-	(16)	a new solution for the same functional requirements with desired quality tradeoffs

For the first objective, we consider several heuristics to identify candidate changes from instable context or quality assumptions in existing feature models. For the second objective, we use the procedure described by an algorithm to analyze for all changes the impacted features and their variability evolution types, on which we base the estimation for the value of the changes. This section first presents the analysis framework, then introduces the steps involved in the process.

4.1. The analysis framework

Our method considers the factors in terms of intentional and contextual changes in the variability evolution analysis. Over a traditional evolving feature analysis, the analysis of the driving factors has an advantage of capturing the trends and influences among requirements, contexts and features.

An overview of the proposed variability evolution analysis framework is depicted in Figure 3. Starting with identifying candidate changes that may cause variability evolution, we analyze their impact on those features on which additional variability will emerge by following the procedure prescribed by Algorithm 1. Specifically, by analyzing the quality and context settings in the problem decomposition structure, the procedure finds those problems whose changed solutions will lead to more variability. Finally, a value-based estimation is conducted to rank all candidate changes. In our method, the value of a change is measured by how much it contributes to the capability of the product line by either involving new applications or improving the customer satisfaction of existing applications. We measure the value of a change from the following three independent dimensions:

1. probability, how likely a change will happen;
2. volume, how many existing and potential applications will be influenced;

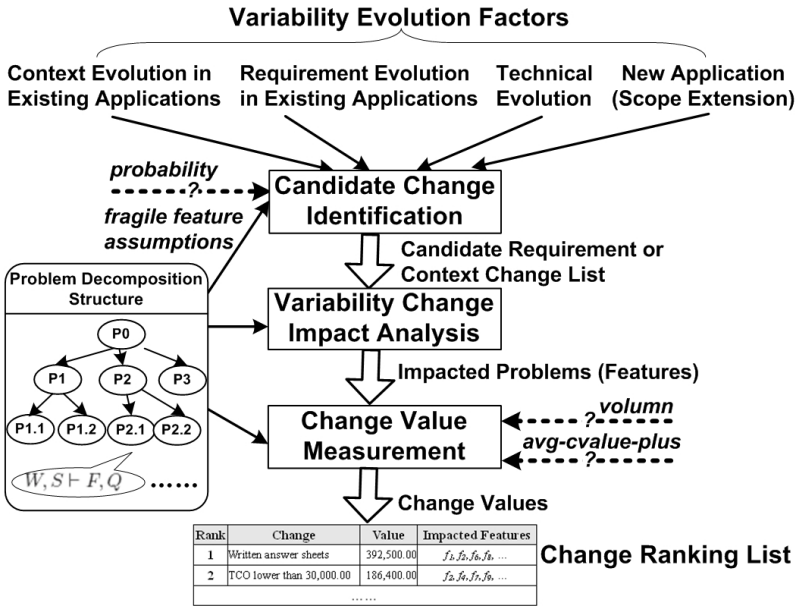


Figure 3. The variability evolution prediction framework

3. value-addition, how much value can be added from the perspective of customers.

Using this analysis framework, one can analyze the past evolution history and predict future variability evolution of an SPL. For an analysis of the evolution history, one shall use pthe ast context or requirement changes instead of possible future changes as the input. Evolution history analysis can help understand the evolution trends and evaluate the past variability decisions. In variability evolution prediction, top-ranked candidate changes can be priorized in the reference architecture design.

4.2. Hierarchical decomposition of problem structures

With those problem decomposition rules presented in Section 3.3, one can construct a fairly complete problem decomposition structure for the targeted domain. The top problem captures the general functional requirement, related quality requirements and the basic context settings. Then it is hierarchically decomposed into a series of AND- or OR sub-problems. Each subproblem captures the solution (feature) for specific functional and quality requirements with certain context assumptions. This problem decomposition structure covers the intentional and contextual diversities among different products by providing alternative sub-problems.

The root problem can be described in the top-level problem diagram, including the general solution to be developed (that can be seen as the root feature), the general functional requirement, related quality requirements and a general context setting. Figure 4 shows the top-level problem diagram for CAGS-PL. We can see that a functional requirement (dotted ellipse) is constrained by a set of quality requirements (cloud shape) of interest to the customers of CAGS-PL. From the extended context diagrams, one can see that the CAGS should load original answers and produce electronic answers for a teacher to grade. Finally, the system will generate reports for students' grades.

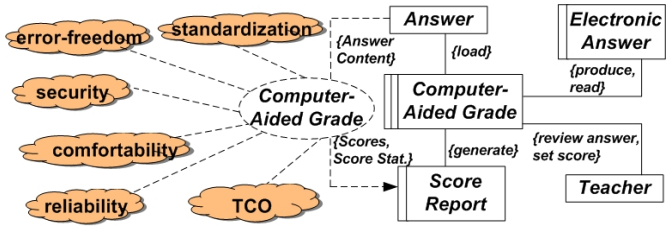


Figure 4. Problem Diagram for the CAGS Root Problem

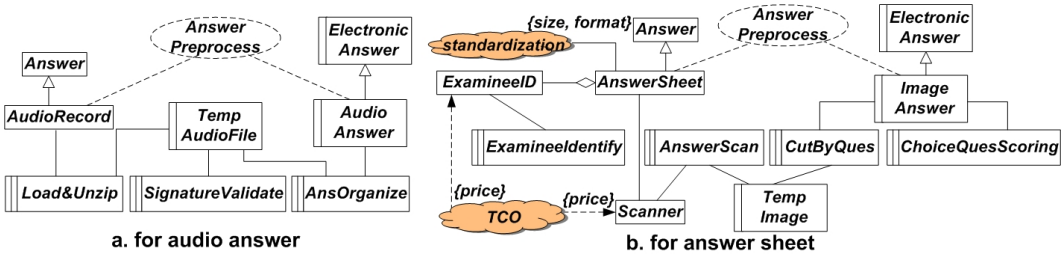


Figure 5. Context-driven Problem Decompositions for Answer Preprocess

Following the decomposition rules and the process described above, we can construct the whole CAGS problem structure by iterative problem decompositions, starting from the root problem shown in Figure 4. Figure 5 and Figure 6 present two examples of OR problem decompositions. The requirement "Answer preprocess" is to transform original answers to electronic answers for the grading purpose. It is a problem relevant to the driving context "Answer". As shown in Figure 5(a) and (b), "Answer preprocess" for AudioRecord in oral exams and for AnswerSheet in written exams are different. Figure 5(b) shows that the quality requirement TCO is sensitive to the price of Scanner, while Standardization is sensitive to the size and format of AnswerSheet. Figure 6 shows that both A3 and A4 answer sheets are applicable to be scanned, however, these two alternative solutions exhibit different quality preferences. The price of an A3 scanner (about 12,000 US dollars) is much higher than that of an A4 scanner (about 1,500 US dollars), whilst "A3 answer sheets" satisfies the standard better than "A4 answer sheets". Therefore, both solutions shall be kept when different preferences are to be considered. In the CAGS domain, customers from smaller middle schools usually prefer the A4 solution for their intra-school exams, and customers of education administration usually prefer the A3 solution for official exams.

Based on the context diagram for each problem, the corresponding problem representations in the canonical form (Rule (3)) can be derived for a further variability evolution analysis.

4.3. Candidate change identification

Changes in assumptions are a primary factor for the evolution of complex systems [23]. One can identify those fragile context or quality assumptions that may change and lead to variability evolution by considering the following heuristic questions:

- *Are there any new alternative driving contexts that may emerge in the near future?*
- *Are there any assumed supporting contexts that cannot be satisfied in all the existing and*

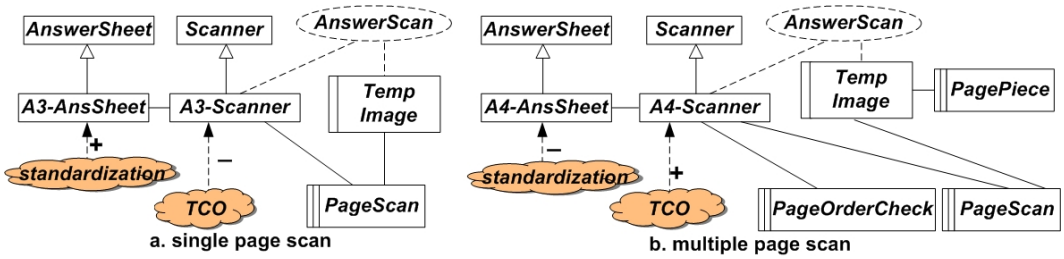


Figure 6. Preference-driven Problem Decompositions for Answer Scan

potential applications?

- *Are there any new future techniques that can promote the product qualities?*
- *Are there any quality tradeoffs that may be unacceptable for potential customers?*

The first two of these questions are about assumptions on contexts, and the last two are about assumptions on quality requirements. Depending on the characteristics of the SPL domains, the estimations for the probability of different kinds of changes vary. The main drivers for such changes are typically scope extension and new technology. The probability of extending the market scope is determined by the market strategy, whilst the probability of adopting new technology is determined mostly by the maturity and availability of the technology.

Considering an existing problem decomposition structure, our variation evolution method deals with a change to contexts or a change to requirements. Such a change is established on specific premises, i.e. the basic assumptions under which the change emerges. For example, the quality change "low TCO" can be associated with context assumptions such as "written exam" and "distributed grading". As a result, each context or requirement change can be represented by a combination of propositions in the conjunctive normal form, involving both the premise and the change itself.

4.4. Variability change impact analysis

After identifying possible context or requirement changes, our evolution analysis should answer the question of how to evolve the feature model to accommodate those changes, i.e. analyze the impacted features of identified changes. As the variation specifications involved in the feature model are based on the current assumptions on the requirements and contexts in the domain, we traverse the problem decomposition structure which hierarchically captures the assumptions on the requirements and contexts for all the features, in order to identify impacted features and their variability evolution types according to those variability evolution rules presented in Section 3.4.

4.4.1. The algorithm

Given a problem decomposition hierarchical structure and a change in the contexts or in quality requirements, Algorithm 1 identifies the necessary changes to the features model accordingly. For example, when the feature "WrittenAnsPreprocess only supports" "A3-sized answer sheets" with the support of an "A3-scanner", the CAGS team considers it necessary to develop new products

```
Input: a problem decomposition structure rooted at r_p, a change c with c.W, c.Q+, c.Q−
Output: influenced problem set INFL with respect to the change
begin
    INFL = {}
    initialize a queue QUEUE
    enqueue the root problem r_p into QUEUE
    while QUEUE is not empty do
        dequeue p from QUEUE with p.W_d, p.W_s, p.S, p.F, p.Q+, p.Q− according to Definition( 5)
        if (c.W ↛ p.W_d) ∨ (p.Q+ ∧ p.Q− ↛ c.Q+ ∧ c.Q−) then
            if p.parent is refined to p using rule DrivingOR or PreferenceOR then
                add p to INFL as "Mandatory/Optional to Alternative" evolution
            end
            else if p.S is not an optional feature then
                add p to INFL as "Mandatory to Optional" evolution
            end
        end
        else if (c.W ↛ p.W_s) then
            add p to INFL as "Mandatory/Optional to Alternative" evolution
        end
        else
            if p is decomposed into subproblems[i], i = 1,...,n using rule AND then
                enqueue subproblems[i], i = 1,...,n into QUEUE
            end
            if p is decomposed into subproblems[i], i = 1,...,n using rule DrivingOR or PreferenceOR then
                branchNotFound = true
                for i = 1,...,n do
                    if (c.W ↦ subproblems[i].W_d) ∧ (subproblems[i].Q+ ∧ subproblems[i].Q− ↦ c.Q+ ∧ c.Q−) then
                        enqueue subproblems[i] into QUEUE
                        branchNotFound = false
                        break
                    end
                end
                if branchNotFound then
                    add p to INFL as "New Variant" evolution
                end
            end
        end
    end
end
```

Algorithm 1: Procedure of identifying influenced feature

for written examinations in middle schools. In this case, the feature "WrittenAnsPreprocess" is required to be extended because it does not satisfy the preference for low TCO currently.

The problem decomposition structure consists of a series of hierarchically structured problems rooted at r_p, and each problem is represented as the definition given below.

Definition 5. *A problem p is a 8-tuple* $< W_d, W_s, S, F, Q^+, Q^-, parent, subproblems >$ *extending the canonical form defined in rule (3) with an additional reference parent to its parent problem and an array of sub-problems. The problem p can be decomposed into a series of sub-problems* $subproblems[i]$ $(i = 1, \ldots, n)$ *using the rule AND, DrivingOR, or PreferenceOR.*

A context or requirement change c can be defined as follows.

Definition 6. *A context or requirement change c is a 3-tuple* $< W, Q^+, Q^- >$*, in which* W*,* Q^+*,* Q^- *satisfying* $W \cup Q^+ \cup Q^- \neq \emptyset$ *are proposition sets representing new emerging context, positively- and negatively-influenced quality requirements respectively.*

For example, the change mentioned above can be represented by the tuple $< W : AnswerSheet \wedge (networkedgrading), Q+ : FS(TCO), Q- : FD(standardization) >$.

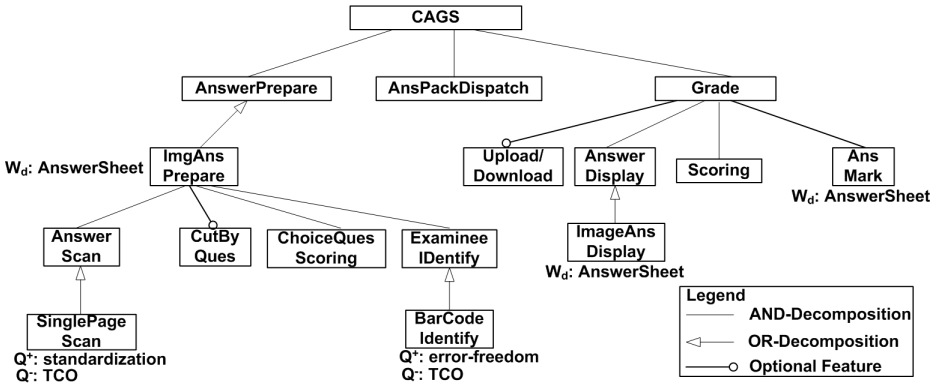


Figure 7. An illustrating problem decomposition structure (feature structure) in CAGS-PL

The algorithm employs a queue to traverse the problem decomposition structure in breadth-first order. For each problem under consideration, the algorithm first evaluates its assumptions according to the change. If its driving context or quality preferences are not consistent with the change, then a *New Variant* evolution will emerge (Line 7-14), determined by whether the refinement between the problem and its parent problem uses AND/OR rules or not. Otherwise, if the required supporting context cannot be satisfied by the change, a *New Alternative* evolution will emerge (Lines 15-17).

If the problem itself is applicable, its sub-problems (if exist) will be considered. If the problem is AND-decomposed, then all its sub-problems will be added into the queue for further evaluation (Lines 19-21), since they are all necessary parts for the problem. If the problem is OR-decomposed, indicating that several alternative sub-problems are applicable, then the algorithm tries to choose any applicable sub-problem, assumptions of which conform to the change, for further evaluation (Lines 22-34). If its solution cannot be found, a new variant solution should be added and a "New Variant" evolution will emerge (Lines 31-33).

4.4.2. An illustrating example

Figure 7 illustrates a problem decomposition structure by annotating a feature model with specific context and requirement settings. Currently, the problem decomposition structure involves four OR-decompositions by driving contexts or by quality preferences. One can see that "ImgAns Prepare", "ImageAns Display" and "AnsMark" are associated with the driving context of "AnswerSheet" for written exams. All of them are common features while "AnswerSheet" is the only form of "Answer". On the other hand,"SinglePageScan" and "BarCodeIdentify" are two features related to specific quality preferences, i.e. "standardization over TCO" and "error-freedom over TCO". They are also common features when these quality tradeoffs are preferred by all the current customers.

With the given problem decomposition structure reflecting the current status of the product line and the changes to be evaluated, Algorithm 1 can be used to automatically identify those problems (features) that may be involved in the variability evolution caused by the changes. Table 2 illustrates two changes to the problem structure in Figure 7 with corresponding variability evolution according to Algorithm 1. The first change introduces AudioAnswer as a new form of answers. The procedure will traverse from the CAGS problem structure, starting from the "AnswerPre-

Table 2. Two changes to the given problem structure and the variability evolution

Change	Impacted Feature	Evolution Type	Description
AudioAnswer	ImgAnsPrepare	Mandatory to Alternative	a new variant of AnswerPrepare for audio answer should be introduced
	ImageAnsDisplay	Mandatory to Alternative	a new variant of audio answer playing should be introduced
	AnsMark	Mandatory to Optional	marking on the answers is meaningless for audio answers
$AnswerSheet \wedge (Q^+ : TCO)$	SinglePageScan	Mandatory to Alternative	a new variant with lower TCO should be introduced with concession to standardization
	BarCodeIdentify	Mandatory to Alternative	a new variant with lower TCO should be introduced with concession to error-freedom

pare" to "ImgAns Prepare", since "AnswerPrepare" only involves general context assumptions that are already implied by the contextual change. When ImgAnsPrepare is analyzed, as its driving context AnswerSheet cannot be implied by the context change and it is an OR refinement to its parent problem by rule (7), it is identified as a candidate of "Mandatory/Optional becomes Alternative" evolution (line 7-8 of the algorithm) and "ImageAnsDisplay" is also identified as a candidate of "Mandatory/Optional becomes Alternative" evolution. Moreover, "AnsMark" is identified as a candidate of "Mandatory becomes Optional" evolution, since it was an AND subproblem of its parent problem. The second change considers another quality tradeoff that prefers TCO over other quality requirements like standardization and error-freedom. This requirement change brings up two candidates of variability evolution.

With influenced features identified by the algorithm, the analyst can further analyze the value of possible changes based on corresponding evolution rules.

4.5. Change value estimation

Having a collection of possible requirement and context changes and their influences, one can then rank them by estimating how much value can be added to the SPL platform when the change is applied. Our value ranking process evaluates the value of a change of contexts or quality requirements using the following metric:

$$\$(c) = P(c) \cdot Vol(c) \cdot \$_{app+}(c) \tag{17}$$

where $\mathtt{P}(c) \in [0, 1]$ represents the probability that the change will emerge, $\mathtt{Vol}(c)$ evaluates the volume by the number of relevant existing or potential products, and $\$_{app+}(\mathtt{c})$ measures the added customer value for each new product.

The change probability can be estimated by analyzing the market and technological trends. For changes driven by technical evolution, the probability can be evaluated by considering the following factors: how probable the new technology will become mature enough and how probable we can well master the technology or find proper technological provider for it in the near future. The volume of the change influence can be estimated by evaluating the existence of the change proposition in all the existing or potential products. For example, for the technological evolution of barcode-based examinee identification, all products of written exams with the preference of error-freedom over TCO are within the scope of the volume.

With the same functional requirements, the customer value for specific customers can be reflected by how well the SPL platform can adapt their specific context to satisfy their quality preferences. We can use the price as the basic value measurement for a product. Then the added value can be computed by multiplying the price and the relative value of all the impacted problems identified in change impact analysis:

$$\$_{app+}(c) = \text{price}_{app} \cdot \sum_{p \in \text{INFL}(c)} \text{weight}(p) \cdot \text{plus\_ratio}(p, c) \tag{18}$$

In the formula, $price_{app}$ represents the average price of relevant application products as the base of customer value measurement, weight(p) denotes the relative weight of a problem to the whole application, and plus_ratio(p, c) evaluates to what ratio the change can improve the satisfaction of the problem.

The weight of each combination of functional and nonfunctional requirements can be evaluated by their sensibility to relevant customers. The plus_ratio can be computed according to different change types:

$$\text{plus\_ratio}(p, c) = \begin{cases} 1 & \text{if } c \text{ is a change to } p.W_d \\ (u_{app}(Q'^{+}/Q'^{-}) - u_{app}(Q^{+}/Q^{-}))/u_{app}(Q^{+}/Q^{-}) & \text{if } c \text{ is a change to } p.W_s \text{ or } p.Q \end{cases} \tag{19}$$

In the equation, Q^{+}/Q^{-} and Q'^{+}/Q'^{-} represent the quality before and after the introduction of the change, u_{app} is the utility function from the perspective of relevant applications (e.g. middle schools that prefer low TCO to standardization). The driving contexts have absolute influence to the satisfaction of functional requirements. While not supported, however, the customer value of the problem will be completely lost. The supporting context change is thus reflected as a remedy to the loss on relevant quality attributes. The added value for newer quality preference is reflected by the promotion of the overall utility function of the quality requirements for relevant customers, that is, answering the question "To what extent the customer satisfaction on the quality of the product can be promoted?"

5. Case study

This section demonstrates the proposed variability evolution analysis method using a case study in the CAGS product line, describing we analysed the history of CAGS-PL during the last several years. The method is evaluated by confirming the benefits by the domain experts if they were using our value-based method.

5.1. The feature model

From the problem decomposition structure described earlier, we can annotate the CAGS feature model in part presented in Figure 8. Each feature has a superscript denoting the phase (see 2.2) in which it is introduced. The variability type for each variation feature is also depicted: gray one is caused by context diversity, diagonal one is caused by different quality preferences. For example, AnswerPrepare and AnswerDisplay are alternative features caused by the driving context Answer, and AnsPackDispatch is caused by the network type (standalone or network-based). ExamineeIDentify is an alternative feature caused by different quality tradeoffs between TCO and

Table 3. CAGS Feature Assumptions

Feature	Feature Assumption
AudioAnsPrepare	AudioRecord, AudioAnswer
ImgAnsPrepare	AnswerSheet, Image Answer, Scanner
SinglePageScan	A3-AnsSheet, A3-Scanner, Standardization+, TCO-
MultiPageScan	A4-AnsSheet, A4-Scanner, Standardization-, TCO+
AnsMark	ImageAnswer
FastForward	comfortability+, error-freedom-

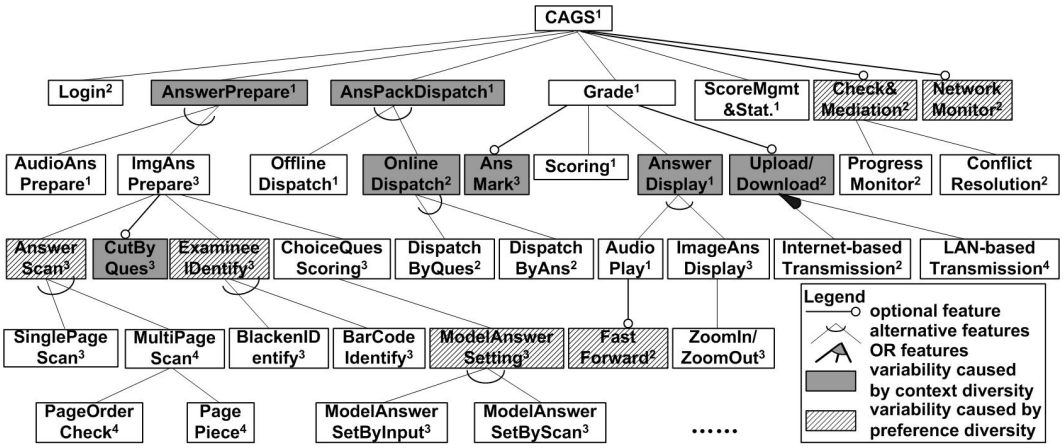


Figure 8. The CAGS Feature Model (Part)

error-freedom: barcode recognition can greatly improve the precision in examinee identification, but needs additional cost.

The variability type for each variation feature is determined through feature assumption analysis. Some of the CAGS feature Assumptions are listed in Table 3, showing that "AudioAns Prepare" and "ImgAns Prepare" are mutually exclusive, because their context assumptions are exclusive: every CAGS application is either for oral or written exams, but never both. Therefore, their parent feature "Answer Prepare" is an alternative feature. The feature "AnsMark" (edit marks and comments on image answer) is optional, because its feature assumption "ImageAnswer" cannot be derived from its parent feature "Grade". The feature "FastForward" is another example of optional feature, which is caused by preference diversity of customers. For some products that satisfy the assumptions of its parent feature "AudioPlay", "FastForward" is not allowed due to its negative effect on the "error-freedom".

5.2. Variability evolution analysis

Following the variability evolution rules, we can analyze those variations introduced in the past several years with an evolutional perspective. The main changes to contexts and requirements after the initial phase are listed in Table 4.

The main changes in the second phase are to quality requirements of error-freedom and reliability and to the context of networked grading. Through a trade-offs analysis, "comfortability"

Table 4. Main changes and influences on the feature variability

W	Phase	Q^+	Q^-	#influenced features
networked grading (W_d)	2			3
networked grading (W_s)	2	error-freedom	comfortability	5
networked grading (W_s)	2	reliability	comfortability	2
networked grading (W_s)	2	comfortability		5
written answer sheet (W_d)	3			5
written answer sheet (W_d)	3	error-freedom	cost	2
written answer sheet (W_d)	3	error-freedom	comfortability	6
written answer sheet (W_d)	3	comfortability		4
A4 sheet and scanner (W_s)	4	cost	standardization	1
intra-school written exam (W_d)	4	comfortability	error-freedom	6

concedes to "error-freedom" and "reliability". During the third phase, the main change is to the contexts of written exams. Error-freedom is still an important quality requirement, but it must be considered within the new context. The fourth phase involves new applications for intra-school written examinations, where cost and comfortability become the main drivers of new variability. Such quality preferences were often achieved by choosing specific contextual domains and their properties, e.g. choosing A4-sized answer sheet and A4-scanner to make a low TCO with the loss of standardization.

Most decisions of variations were made in a reactive mode, i.e. by adapting the product line when changes emerge. Through interviews with the CAGS-PL team, we found several deficiencies in the decisions made in the past. For example, as required by supporting written examinations during the third phase, it took the developers greater effort to modify the features to adapt audio handling to answer sheets image handling. If the change had been foreseen and properly considered in the design of the product, it would have been much easier to introduce this new variation of audio handling to handle images.

5.3. Value-based estimation in variability evolution

Due to confidentiality, here we only present the value evaluation of two cases in variability evolution. At the third phase, we compared two candidate changes: A4-sized answer sheet (Change 1) and barcode for examinee ID (Change 2). The Change 1 is introduced because of the preference of "low TCO", but we knew that breaks the quality goal of "standardization". The Change 2 is a change driven by technological development for a better precision in examinee recognition, which can be seen as a sub-goal to fulfil the parent goal of "error-freedom".

Through interviews with the market team, we got the data for evaluation as listed in Table 5. From the results of value evaluation, one can see Change 1 is much more valuable to be involved than the Change 2. Therefore, Change 1 should be considered as having priority with respect to Change 2. This result has been recognized by the CAGS team. Restrospectively, however, they actually introduced the Change 2 at the third phase and the Change 1 at the fourth phase. This resulted in the subsequent refactoring to scan-related components due to the lack of separation of commonality and variability.

Table 5. Value evaluation for two past changes

Data	Change 1	Change 2
probability	At that time, the probability of entering the market of middle school was estimated to be 70%.	At that time, the probability of getting mature barcode recognition techniques was estimated to be 80%.
influenced features	AnswerScan	ExamineeIdentify
volume	There are totally about 2,000 middle schools in the local region, the anticipated market share is 30%, and 40% of them prefer low TCO to standardization, making a volume of 240=2000*30%*40%.	There are totally about 40 district-level education administrations in local and near regions, the anticipated market share is 50% and 90% of them prefer precision to low TCO, making a volume of 18=40*50%*90%.
price	The average price of CAGS products for middle schools is 4,300 US dollars.	The average price of CAGS products for administrations is 14,300 US dollars.
weight	The relative weight of TCO/standardization is evaluated to 0.3.	The relative weight of precision in examinee recognition is evaluated to 0.1.
plus-ratio	The cost on scanner is reduced from 12,000 to 1,500 US dollars, but the goal of standardization is broken. The new tradeoff makes a total plus ratio of 70%.	With barcode recognition, the precision of examinee recognition can be improved from 95% to 99.99% at the additional cost. The new tradeoff makes a total plus ratio of 60%.
value	70%*240*30000*0.3*70% =1,058,400	80%*18*100000*0.1*60% =86,400

6. Evaluation and Discussion

In this section, we first evaluate the benefits of the proposed problem-oriented analysis method for variability evolution in SPL development and maintenance, then we discuss the completeness and correctness of the method.

6.1. Evaluation

If successfully applied, the proposed problem analysis provides the intentional and contextual basis for feature variations thus facilitates variability analysis and captures the rationales by intentional and contextual propositions. Thus, product customization and variability evolution can also be eased.

Help capturing the rationale of variability In the segment of feature model in Figure 8, there are 6 optional features, 7 alternative features and 1 OR feature. Because "Upload/Download" is both optional an OR subfeature, these amounts to 13 distinct variant features. The context and preference factors that cause these variation features are listed in Table 6, showing that all the variation features can be interpreted by 3 contextual diversities and 5 preferencial diversities. Considering the relevance among them, those quality preferences can be further classified as high-end (preferring error-freedom, security, etc) and low-end (preferring comfortability, TCO, etc) with different weights.

Reducing decision making effort in customization Product customization using a feature model helps derive the product feature model from the domain feature model [21]. With the intentional and contextual feature assumptions captured in problem analysis, we can derive the product model by evaluating the feature assumptions. As listed in Table 6, we can derive a basic product model by only considering 3 context factors and 1 quality preference (either high-end or low-end), instead of went through all the 13 variant features individually. Thus, the efforts needed

Table 6. Variation Features Caused by Each Context or Preference Diversity

Factor	Type	Variation Features
Answer (audio or written)	Context	AnswerPrepare, AnsMark, AnswerDisplay (3)
Grading Mode (networked or stand-alone)	Context	AnsPackDispatch, Upload/Download, CutByQues (3)
Dispatch Mode (by answer or by question)	Context	OnlineDispatch (1)
comfortability/error-freedom	Preference	Check&Mediation, FastForward (2)
comfortability/error-freedom&security	Preference	ModelAnswerSetting (1)
comfortability/reliability	Preference	NetworkMonitor (1)
TCO/standardization	Preference	AnswerScan (1)
TCO/error-freedom	Preference	ExamineeIDentify (1)

for variability communications with the customers and the customization decision-making can be reduced greatly. This embodies the principle emphasized by Pohl et al. [22] and others: explicitly document variability rationales and use them to help customers and domain engineers to make decisions on variant choices. Furthermore, this kind of rationale-based variability customization can better ensure the consistency of the variability configuration, since most of the variability constraints come from the relevance of intentional and contextual settings.

Identifying reasons for introducing new variability In the long-term SPL life cycle, variations continuously evolve along with scope extensions and changes in existing products. Feature assumptions represent the current intentional and contextual cognitions on the SPL domain, determining the current feature variations. By analyzing how the intentional or contextual changes affect existing feature assumptions, one can precisely identify the reason for introducing new variability. For example, CAGS-PL has evolved for 6 years and 4 phases, and the superscript of each feature in Figure 8 represents the phase in which the feature was introduced. In the Phase 3, the contextual setting of "AnswerSheet" was introduced, then variability evolution might occur on all the features with the assumption of "AudioRecord", e.g. both "AnswerPrepare" and "AnswerDisplay" evolve from mandatory features into alternative features. With this kind of variability evolution analysis, one can better understand the evolution history. Moreover, one can further predict future variability changes by revealing the fragile feature assumptions.

6.2. Discussion

In this paper, we identify a series of variability evolution patterns by rules (11) to (16) in terms of different context or requirement changes. We are not trying to cover all possible changes in a small set of rules, but to consider them as elementary patterns that our case study has found to be relevant. The changes in reality may be much more complex, but they can be analyzed as a combination of these rules under the following assumptions. First, the root problem, e.g. the CAGS root problem shown in Figure 4, is shared by all the products and does not change in SPL evolution. This means that the general functional and nonfunctional requirements of different products must not change in SPL evolution. What differs among products are contextual and requirement diversity which evolves with the features derived from problem decomposition. Second, the context and requirement diversity among products always increases monotonically, which means that existing differences among the contexts and requirements of different products

will not disappear in the future as all rationales are kept. Third, as a simplification we do not consider the changes of dependencies among contexts and requirements, e.g. exclusive context settings will always be exclusive. Given the above assumptions and simplifications, variability evolution can always be analyzed as variability increments to the current problem decomposition structure, following the procedure described in Algorithm 1.

Of course, variability evolution in real SPL engineering is much more complex due to the occurrence of changes beyond our assumptions. (i) These complex evolution will cause global reconstructions to the problem decomposition structure. In CAGS-PL, for example, new general requirements about "improvement by error learning" emerge and require a series of new features like error recording, classifying, analysis and history management for each examinee to be introduced. (ii) Context and requirement diversity can shrink, making feature variation decrease due to market shrink or changes in existing products. For example, if the CAGS team decides to quit the low-end market or all the low-end products change to adopt standard A3-sized answer sheets, the diversity between A3- and A4-sized answer sheets and corresponding answer preparation solutions will disappear. (iii) Example of complex changes of dependencies among contexts and requirements can be the exclusion relation between "ImageAnswer" and "AudioAnswer". If some products in the future involve both oral and written questions together in the same examination at the runtime, the two context statements of ImageAnswer and AudioAnswer will evolve to be non-exclusive. This change of context dependency will turn the alternative feature "AnswerPrepare" to become a runtime OR-decomposed feature. These complex evolution types require new problem decomposition for new general requirements and possible reconstructions to existing problem decomposition rather than just introducing new variation branches. It is our future work to investigate how the rules will adapt to accommodate such runtime changes, as these complex variability evolution patterns are not captured by Rules (11) to (16) and not analyzed using Algorithm 1.

7. Related work

We review the literature from both the requirements engineering and the software product line engineering fields on the following three related subjects.

Domain requirements engineering. Existing domain analysis methods [9, 17, 22] observe and analyze domain-wide commonality and variability from a static perspective. The process of eliciting the change of variability from the stakeholders is largely absent. However, in practical SPL development, proactive and reactive reuse are usually interweaved and variations are introduced gradually as a result. Therefore, analyzing and understanding the variability change in an evolutionary perspective is necessary. In this paper, we focus on an analysis of evolving requirements and contexts in order to understand the development history and to predict potential variations in features.

Recent work tries to combine problem analysis with feature models and variability analysis: Classen et al. [5, 6] use the Problem Frames approach to analyze the physical environment of the feature to establish mappings between features and problem frames and achieve a deeper understanding of the variability. Salifu et al. [24] on the other hand, analyze the variability native in variant frames in the context of SPL. Ali et al. [2] present several ways by which the context can

be related to requirements and subsequently used for product derivation. Yu et al. [29] attempt to configure features using high-variability goal models. However, these work do not provide a systematic way to provide the rationale for the changes in variability evolution.

Volatility requirements. Requirements are involved in software evolution and change for most software systems [7]. Therefore, some evolution analysis techniques have been proposed to interpret the motivations behind changes, in order to better understand the change history and even to predict the trend of changes. For example, volatile requirements are considered concerns that are likely to change during development or after operation. Moreira et al. propose to model volatile requirements with early aspects techniques [18]. These works focus on the modeling and specification of volatile requirements only, and it is not explicit how to discover them. Savolainen et al. [25] propose a volatility analysis framework to estimate evolution from two perspectives: the probability of a change in requirement in the future, and vagueness of the requirement. However, their work does not recognise the influences of context diversity in variability evolution.

Variability and evolution management. Variability management includes the activities of variability capturing, modeling, implementation, traceability and also evolution. Svahnberg et al. [26] presents an experience report on variability evolution and the connections between evolution and variability. Their work focuses on the evolution of SPL architecture and components for requirement evolution, however, the analysis and prediction of requirement variability are not made explicit. Thurimella et al. [27] propose a rationale-based SPL evolution method using the Questions, Options and Criteria model [16] and a modified version of EasyWinWin [3]. Although it involves a value-based viewpoint on SPL evolution, there is no formal basis for variability evolution.

8. Conclusion

Effective variability evolution management is the premise to achieve successful proactive reuse. By interpreting and recognizing the variation evolution trend, the understanding on how the SPL requirements have evolved can be improved greatly. By foreseeing variation evolution, an SPL can be instrumented with appropriate variability mechanisms accordingly [26] [25]. Thus, the time and cost needed in the reactive SPL refactoring and extensions can be reduced. In this paper we establish a canonical form for tracing possible variations of a software product line features to the changes in requirements and contexts. Classifying changes to contexts by their impact on functional and quality requirements, this canonical form is used to separate changes of driving contexts from those of quality supporting contexts. Such a detailed analysis enables us to further assess the value of adding variations to the SPL feature models for potential changes, taking into account the probability, the change impact and the range of changes. The rules for variability evolution and the impact analysis algorithm help one to identify places on the feature models for candidate variations, after which the computation of added-values further help one to decide on extending the variability of feature models. We have applied the above analysis framework to a commercial SPL case study and report the feedbacks from the domain engineers at different phases of its maintenance activities.

Acknowledgments. We would like to thank the anonymous reviewers for their valuable comments and suggestions that allowed us to improve our paper. This work is supported by National Natural Science Foundation of China under Grant No. 90818009, National High Technology

Development 863 Program of China under Grant No. 2009AA010307, EU FP7 SecureChange, and Shanghai Committee of Science and Technology, China under Grant Nos. 08DZ2271800, 09DZ2272800.

References

[1] Raian Ali, Fabiano Dalpiaz, and Paolo Giorgini. A goal-based framework for contextual requirements modeling and analysis. *Requir. Eng.*, 15(4):439–458, 2010.

[2] Raian Ali, Yijun Yu, Ruzanna Chitchyan, and Paolo Giorgini. Towards eliciting contextual variability in requirements. In *Proceedings of the 4th International Workshop on Software Product Management (IWSPM '09), collocated at the Requirements Engineering conference (RE'09)*. IEEE Computer Society, 2009.

[3] B. Boehm, P. Grünbacher, and R. Briggs. Developing groupware for requirements negotiation: Lessons learned. *IEEE Software*, 18(3), 2001.

[4] Barry Boehm and Apurva Jain. An initial theory of value-based software engineering. In *Value-Based Software Engineering*. Springer Verlag, 2005.

[5] A. Classen, P. Heymans, R. Laney, B. Nuseibeh, and T. Tun. On the structure of problem variability: From feature diagrams to problem frames. In *VaMoS '07*, pages 109–117, 2007.

[6] A. Classen, P. Heymans, and P. Schobbens. What's in a feature: A requirements engineering perspective. In *FASE '08*, pages 16–30, 2008.

[7] M. Dambros. Supporting software evolution analysis with historical dependencies and defect information. In *ICSM '08*, 2008.

[8] A. Dardenne, A. Lamsweerde, and S. Fickas. Goal-directed requirements acquisition. *Sci. Comput. Program.*, 20(1-2):3–50, 1993.

[9] K. Kang et al. FORM: a feature-oriented reuse method with domain-specific reference architectures. *Annals of Soft. Eng.*, 5:143–168, January 1998.

[10] J. Hall, L. Rapanotti, and M. Jackson. Problem frame semantics for software development. *Software and System Modeling*, 4(2):189–198, 2005.

[11] J. Hall, L. Rapanotti, and M. Jackson. Problem oriented software engineering: Solving the package router control problem. *TSE*, 34(2):226–241, 2008.

[12] M. Jackson. *Problem Frames: Analysing and Structuring Software Development Problems*. Addison-Wesley, February 2001.

[13] I. Jureta, J. Mylopoulos, and S. Faulkner. Revisiting the core ontology and problem in requirements engineering. In *RE '08*, pages 71–80. IEEE Computer Society, 2008.

[14] S. Liaskos, L. Jiang, A. Lapouchnian, Y. Wang, Y. Yu, J. Leite, and J. Mylopoulos. Exploring the dimensions of variability: a requirements engineering perspective. In *VaMoS '07*, pages 17–26, 2007.

[15] S. Liaskos, A. Lapouchnian, Y. Yu, E. Yu, and J. Mylopoulos. On goal-based variability acquisition and analysis. In *RE '06*, pages 76–85. IEEE Computer Society, 2006.

[16] A. MacLean, R. Young, V. Bellotti, and T. Moran. Questions, options, and criteria: Elements of design space analysis. *Human-Computer Interaction*, 6(3):201–250, 1991.

[17] M. Mikyeong, Y. Keunhyuk, and S. Heung. An approach to developing domain requirements as a core asset based on commonality and variability analysis in a product line. *TSE*, 31(7):551–569, 2005.

[18] A. Moreira, João Araújo, and J. Whittle. Modeling volatile concerns as aspects. In *CAiSE '06*, pages 544–558, 2006.

[19] J. Mylopoulos, L. Chung, and B. Nixon. Representing and using nonfunctional requirements: A process-oriented approach. *TSE*, 18(6):483–497, 1992.

[20] X. Peng, S. Lee, and W. Zhao. Feature-oriented nonfunctional requirements analysis for software product line. *Journal of Computer Science and Technology*, 24(2):319–338, 2009.

[21] X. Peng, W. Zhao, Y. Xue, and Y. Wu. Ontology-based feature modeling and application-oriented tailoring. In *ICSR '06*, 2006.

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

[23] B. Ramesh and M. Jarke. Toward reference models of requirements traceability. *TSE*, 27(1):58–93, 2001.

[24] M. Salifu, B. Nuseibeh, L. Rapanotti, and T. Tun. Using problem descriptions to represent variabilities for context-aware applications. In *VaMoS '07*, pages 149–156, 2007.

[25] J. Savolainen and J. Kuusela. Volatility analysis framework for product lines. *Softw. Eng. Notes*, 26(3):133–141, 2001.

[26] M. Svahnberg. Variability in evolving software product lines. Technical report, Blekinge Institute of Technology, Department of Software Engineering and Computer Science, Karlskrona, Sweden, 2000.

[27] A. Thurimella and B. Bruegge. Evolution in product line requirements engineering: A rationale management approach. In *RE '07*, pages 254–257. IEEE, 2007.

[28] Y. Yu, A. Lapouchnian, S. Liaskos, J. Mylopoulos, and J.C.S.P. Leite. From goals to high-variability software design. In *Proc. 17th Int. Symposium on Methodologies for Intelligent Systems (ISMIS), Toronto, Canada, May 20-23*, pages 1–16, 2008.

[29] Y. Yu, J. Leite, A. Lapouchnian, and J. Mylopoulos. Configuring features with stakeholder goals. In *SAC '08*, pages 645–649, 2008.

[30] P. Zave and M. Jackson. Four dark corners of requirements engineering. *TOSEM*, 6(1):1–30, 1997.