Quality Attributes Modeling in Feature Models and Feature Model Validation in Software Product Lines

A thesis submitted in fulfillment of requirement for the degree of

Doctor of Philosophy

GUOHENG ZHANG

School of Electrical Engineering and Computer Science
The University of Newcastle,
Callaghan, NSW, 2308, Australia

March 2013
Statement of Originality

The thesis contains no material which has been accepted for the award of any other degree or diploma in any university or other tertiary institution and, to the best of my knowledge and belief, contains no material previously published or written by another person, except where due reference has been made in the text. I give consent to the final version of my thesis being made available worldwide when deposited in the University's Digital Repository.

Guoheng Zhang
Acknowledgement

I would like to acknowledge the support and encouragement of several people, without whom, this thesis would not be possible.

Firstly, I am grateful for my two supervisors, Associate Prof. Huilin Ye and Dr. Yuqing Lin. I have learnt almost everything about research from them. In the last four years, they have given me guidance and encouragement, support and generosity in sharing their time, knowledge, and expertise during the preparation of my Ph.D thesis and throughout my research work.

My research has been funded by Australian Research Council under Discovery Project DP0772799. I would like to thank the Council for this great opportunity.

I wish to thank my wife for her love, encouragement and continuous support. I am grateful to my parents for their support and education.

During several conferences, I had the pleasure to meet many passionate people from the software product line community. I wish to give my thanks to David Benavides, Xin Peng and Wei Zhang in sharing with me their ideas in SPLC 2011 and RE 2010.
# Table of Contents

Acknowledgement ........................................................................................................... ii

Table of Contents .............................................................................................................. iii

Abstract ............................................................................................................................. vii

1. **Introduction** .............................................................................................................. 1
   1.1 Problem Statement ............................................................................................ 3
   1.2 Proposed Approach ......................................................................................... 6
   1.3 Contributions .................................................................................................. 8
   1.4 Thesis Structure .............................................................................................. 12
   1.5 Publication List ............................................................................................... 13

2. **Background** ............................................................................................................ 15
   2.1 Software Product Lines ................................................................................ 15
   2.2 Feature Models ............................................................................................... 21
      2.2.1 Feature Diagram and Product Configuration ....................................... 22
      2.2.2 Two Problems ....................................................................................... 28
   2.3 Feature Model Validation ............................................................................. 30
      2.3.1 Feature Model Errors .......................................................................... 30
      2.3.2 Related Works ..................................................................................... 33
   2.4 Quality Assessment in Product Configuration ............................................ 35
      2.4.1 Quality Attributes Modeling in Feature Models ................................ 35
      2.4.2 Related Works ..................................................................................... 37
   2.5 Summary ........................................................................................................... 41

3. **An Efficient Approach for Feature Model Validation** ......................................... 43
   3.1 Causes of Feature Model Errors .................................................................. 43
   3.2 Feature Relationship Propagation ................................................................ 46
   3.3 Detecting Feature Model Errors ................................................................... 56
   3.4 Explaining Feature Model Errors ................................................................. 64
   3.5 Concluding Remarks .................................................................................... 68

4. **An Approach for Quality Attributes Modeling in a Feature Model** ...................... 71
4.1 Overview ........................................................................................................................................... 71
4.2 Identify and Represent Quality Attributes in Feature Models ....................................................... 73
4.3 Measure Interdependencies between Features and Quality Attributes .............................................. 80
   4.3.1 Identifying Contributors ................................................................................................................ 83
   4.3.2 Prioritizing Contributors ............................................................................................................... 85
   4.3.3 Identifying Feature Relationships ................................................................................................. 88
   4.3.4 Calculating the Overall Impact ..................................................................................................... 90
   4.3.5 Normalizing the Overall Impact .................................................................................................... 92
   4.3.6 Evaluating Domain Experts’ Judgments ....................................................................................... 95
   4.3.7 Representing the Interdependencies ............................................................................................. 101
   4.3.8 Managing Relationships among Quality Attributes ..................................................................... 106
4.4 Quality-Aware Product Configuration ............................................................................................. 109
   4.4.1 Validating Quality Requirements ................................................................................................. 111
   4.4.2 Making Decisions ......................................................................................................................... 113
   4.4.3 Modifying Configured Products .................................................................................................. 118
4.5 Concluding Remarks ....................................................................................................................... 121

5. Prototype Tool ....................................................................................................................................... 123
   5.1 Feature Model Editor Tool ............................................................................................................... 123
   5.2 Feature Model Validation Tool ....................................................................................................... 126
   5.3 Quality Attributes Modeling Tool .................................................................................................. 128
   5.4 Product Configuration Tool .......................................................................................................... 132
   5.5 Concluding Remarks ..................................................................................................................... 135

6. Evaluation ............................................................................................................................................ 136
   6.1 Evaluation of Feature Model Validation ......................................................................................... 136
   6.2 Evaluation of Quality Assessment ................................................................................................ 142

7. Conclusion ........................................................................................................................................... 156
   7.1 Summary ........................................................................................................................................ 156
   7.2 Future Work .................................................................................................................................... 159
Figure 1.1 The Proposed Framework ................................................................. 7
Figure 2.1 Framework of Software Product Line Engineering.......................... 17
Figure 2.2 A Feature Model of Tourist Guide System Software Product Line ........ 25
Figure 2.3 Examples of Feature Model Errors .................................................. 31
Figure 3.1 Examples of Feature Model Errors (same with Figure 2.3) ................. 44
Figure 3.2 Transferred Feature Model of Tourist Guide Software Product Line ........ 51
Figure 3.3 A Feature Model for Illustrating Feature Relationship Propagation .......... 56
Figure 3.4 An Example for Illustrating Minimal Explanations ............................ 67
Figure 4.1 Overview of Modeling Quality Attributes in Feature Models ................ 73
Figure 4.2 A Softgoal Model of Banking System ............................................. 75
Figure 4.3 A Portion of Sort Catalogue of Security and Performance .................. 76
Figure 4.4 A Portion of Operationalization Method Catalogue of Security ............. 77
Figure 4.5 The Extended Feature Model of Tourist Guide Software Product Line ..... 80
Figure 4.6 The Comparison Matrix of Contributors of Data Transfer Speed ........... 88
Figure 4.7 The Relative Importance Values of Individual Contributors of Data Transfer Speed. 88
Figure 4.8 A Feature Diagram for the Contributors of Data Transfer Speed ........... 93
Figure 4.9 The Representation Schema for Interdependency ............................. 102
Figure 4.10 The Representation Schema for Data Transfer Speed ....................... 104
Figure 4.11 Process of Quality-Aware Product Configuration ............................. 110
Figure 4.12 Variation Point View of Tourist Guide SPL Feature Model ................. 118
Figure 5.1 Function View of Editing Feature Models in FMETool ....................... 124
Figure 5.2 Function View of Generating Feature Models in FMETool ................... 125
Figure 5.3 XML Format for Representing Feature Models .................................. 126
Figure 5.4 Function View of Validating Feature Models in FMVTool .................... 128
Figure 5.5 Function View of Loading Feature Models in QAMTool ..................... 129
Figure 5.6 Function View of Identifying Contributors in QAMTool ..................... 130
Figure 5.7 Function View of Prioritizing Contributors in QAMTool ..................... 131
Figure 5.8 Function View of Validating Quality Requirements in QAPCTool .......... 133
Figure 5.9 Function View of Making Feature Selections in QAPCTool .................. 134
Figure 5.10 Function View of Modifying Product Configuration ......................... 135
Figure 6.1 The Experiment for Evaluating Feature Model Validation Approach ........ 137
Figure 6.2 An Example of Random Feature Model .......................................... 141
Figure 6.3 The Experiment for Evaluating Quality Assessment Approach .............. 143
Figure 6.4 Functionalities of a CAD System for Police ................................... 144
Figure 6.5 Feature Model of CAD Software Product Line (1) ............................... 145
Figure 6.6 Feature Model of CAD Software Product Line (2) ............................... 146
Figure 6.7 Quality Attribute Feature Tree of CAD Software Product Line ............. 147
Figure 6.8 Interdependency of Ease of Use ..................................................... 148
Figure 6.9 Interdependency of Data Access Security ....................................... 149
Figure 6.10 Interdependency of Dispatch Respond Time ................................... 149
Figure 6.11 Interdependency of MLC, DTS and DTSS ..................................... 152
Table 2.1 The Notations and Semantics of Feature Relationships in CBFM.........................24
Table 2.2 Matrix for Maintaining Constraints in Tourist Guide Software Product Line...........26
Table 3.1 Illustration of Adjacent Features in Tourist Guide Software Product Line ..........50
Table 3.2 Examples of Generating Minimal Explanations..................................................68
Table 4.1 Intensity of Importance in Analytic Hierarchical Process....................................86
Table 4.2 Illustration of Calculating OIV (DTS, VS) .........................................................92
Table 4.3 Valid Selections for Data Transfer Speed in Tourist Guide Software Product Line ......94
Table 4.4 Comparison Matrix for the Contributors of QA..................................................96
Table 4.5 Quality Attribute Knowledge Base Tables.........................................................107
Table 4.6 The QAKB Table of Data Transfer Speed and Data Transfer Security...................109
Table 4.7 Product Configuration Process in Feature Model of Tourist Guide SPL................117
Table 4.8 Modification Solutions .......................................................................................121
Table 6.1 Validation Results from FAMA and FMVTool on Pre-designed Feature Models.......138
Table 6.2 Validation Results from FMVTool on Randomly Generated Feature Models........140
Table 6.3 Time Spent for Identifying Feature Model Errors by FMVTool and FAMA ..........142
Table 6.4 Contributors of DAS, EOU and DRT .................................................................150
Table 6.5 Configured Products from Feature Model of CAD Software Product Line ..........150
Table 6.6 Comparison between Quality Assessment Approach and Testing Domain Expert ...151
Abstract

In a software product line, a feature model represents the commonalities and variabilities among a family of software systems. Each valid combination of features authorized by a feature model corresponds to a possible product of the software product line. In feature-based product configuration, the desired features are selected from a feature model based on the customers’ requirements, but the selected features must satisfy the selection constraints specified in the feature model.

In practice, two problems arise as the major obstacles of using feature models in product configuration. Firstly, a feature model may have errors which must be resolved for the effective product configuration. The feature model validation aims to identify the feature model errors and provide explanations for each identified error. The current validation approaches transform a feature model into a constraint satisfaction problem (CSP) and use solvers to reason on the CSP. However, the use of solvers might take an infeasible amount of time for validating large-scale feature models, as CSP exhibits the exponential complexity and requires a combination of heuristics and combinational search methods. A more efficient feature model validation approach is needed. Another problem is to assess the product quality in feature-based product configuration. In software development, the product quality is mostly handled until the final product is generated and tested. However, if the final product cannot satisfy the customers' quality requirements, it will be very costly to fix the problems. Therefore, the product quality should be assessed in feature-based product configuration which is considered as the first stage of deriving valid products. To achieve this aim, the quality attributes must be modelled in a
feature model. The current quality attributes modeling approaches have several limitations, such as requiring real products which are difficult to obtain or involving onerous human efforts. A systematic quality attributes modeling approach is needed to reduce the efforts involved in domain experts’ judgments.

This research aims to address the above problems. First, we developed an efficient validation approach based on the contradictory feature relationships behind the errors. As the contradictory feature relationships were found based on feature relationship propagation, the solvers were not required by this approach. Second, we developed a quality attributes modeling approach which uses domain experts to make judgments and uses analytic hierarchical process to reduce the efforts involved in the judgments. A knowledge database called quality attribute knowledge base is generated to maintain the captured quality knowledge. Finally, we enhanced the feature-based product configuration with the captured quality knowledge. A desired product is configured from a feature model in a quality-aware manner. A prototype tool was developed to implement the concepts of the proposed approaches and a set of empirical experiments were carried out to evaluate the proposed approaches.
1. Introduction

Cost, quality and time-to market have been the main concerns in software engineering since the 1960s (Sinnema and Deelstra 2006). Over the last decade, the software product line (SPL) approach emerged as an important and growing software development paradigm to help software companies gain potential advantages in terms of the above concerns (Pohl et al. 2005).

In an SPL approach, a common infrastructure is first established based on the exploited commonalities and variabilities among a family of software systems in a particular domain and then new software products are developed by reusing the common infrastructure (Clements and Northrop 2002). Compared with traditional software reuse, the SPL approach enables more systematic software reuse. The reuse process is more strategic and the reusable parts can be derived from any stage of software development life cycles. Currently, the research field of SPL is very active and the researchers in the software product line community have made great achievements in the areas of modeling, customizing and realizing software product lines.

The SPL approach starts from domain engineering which is the process of establishing a software product line based on domain requirements analysis. In the last decade, feature models were widely used in an SPL approach to represent the knowledge captured in domain engineering. The commonalities and variabilities among the family of software systems in a domain are explicitly represented in a feature model (Czarnecki 1998). Mostly, a feature model is represented by an AND/OR tree-structure feature diagram where nodes represent the features and edges represent the relationships among features. In a feature model, the features describe
the distinguishable characteristics of software systems in an SPL (Czarnecki and Eisenecker 2000) and the feature relationships constrain the way in which features can be combined into specific products (Mendonca 2009). In literature, an ever growing number of approaches have been proposed to define the notations of feature models (Benavides et al. 2005; Czarnecki et al. 2005; Kang et al. 1990; Riebisch et al. 2002) and use feature models in an SPL approach in different ways (Eriksson et al. 2005; Griss et al. 1998; Kang et al. 1998).

Although feature models can be used in different aspects of an SPL approach, one important use is to prevent the derivation of illegal product specifications during product configuration. The process of legally selecting the desired features from a feature model based on the customers’ requirements and the selection constraints specified in the feature model is called feature-based product configuration (FBPC). Each valid combination of features derived from FBPC corresponds to a possible product of the software product line. The FBPC can be time-consuming and error-prone due to the complexity caused by the large number of features and feature relationships. A set of approaches have been proposed to improve the FBPC in different ways, such as providing automated support (White and Schmidt 2008; White et al. 2009), optimizing configuration problem (Loesch and Ploedereder 2007; Mendonca et al. 2008; Segura 2008) and developing supporting tools (Antkiweica and Czarnecki 2004; Abbasi et al. 2011; Botterweck et al. 2009a, 2009b; Rabiser et al. 2007). Although impressive steps have been taken for FBPC, some problems will prevent applying the feature-based product configuration in practice. The following section will present these problems and explain why they are not addressed properly.
1.1 Problem Statement

Firstly, a feature model may contain errors which will prevent the effective product configuration (Batory et al. 2006; Benavides et al. 2010). For example, a feature model is invalid or void if no products can be derived from the feature model. A feature model is incorrect if it contains features that cannot be part of any product or it contains variable features that will appear in all products. To correct an invalid feature model, the feature modelers need to revisit the feature model and modify or eliminate the right set of feature relationships. As it is a time-consuming and error-prone task for the feature modelers to manually correct the feature models, an automated validation approach is very important. The current most popular automated validation approaches transform a feature model into a constraint satisfaction problem (CSP) or Boolean satisfaction problem (SAT) and use the off-the-shelf solvers to reason on the CSP or SAT (Czarnecki and Kim 2005; Trinidad et al. 2008; Zhang et al. 2004). The feature model errors and their corresponding explanations can be identified by finding the valid solutions from CSP or SAT under certain conditions.

Despite the current validation approaches having gained relative success by making use of solvers to provide the automated support, they have two main limitations. First, it is well known that the use of solvers can lead to state space explosion problem, as CSP or SAT exhibits the exponential complexity and requires a combination of heuristics and combinational search methods (Mackworth 1977; Segura 2008; Yan et al. 2009). This means that a solver might take an infeasible length of time to validate large-scale feature models, which is not acceptable in practice. Second, the current validation approaches debug a feature model error by returning all
the possible solutions of fixing the error (also called error explanations). Each error explanation is a right set of feature relationships that must be modified to remove the error (Trinidad et al. 2008). Although the feature modelers can resolve a feature model error based on a specific error explanation, they are not aware of the cause of the error (conflicting feature relationships). Then the feature model modification based on a specific error explanation may introduce new errors into the feature model. Therefore, two research questions can be raised with respect to feature model validation.

1. **How to develop a feature model validation approach that is more efficient than the current CSP-based approaches.**

2. **How to find a better way to explain the feature model errors.**

Once all errors have been removed from a feature model, products can be configured from the validated feature model. The functionalities of a target product can be realized by selecting the desired features from a feature model while the product quality is mostly handled until the final product is generated and tested (Montagud and Abrahao 2009). However, if the generated product cannot satisfy the customers’ quality requirements at the latter stages of product development; it would be very costly to fix the problems as the customers would need to return to the beginning of the development of the life cycles to re-configure the product. Therefore the estimation and prediction on the product quality should be considered as early as possible. As feature-based product configuration is considered as the first stage of preventing deriving
invalid products in product development, the product quality should be assessed at this stage. To achieve this aim, the quality attributes must be modelled in a feature model.

The key issue of modeling quality attributes in a feature model is to understand the interdependencies between selected features and product quality, such as measuring how the selection of individual features will affect a quality attribute and how the combination of a set of features will affect the quality attribute. In literature, some approaches use the feedback strategy to measure the interdependencies between features and quality attributes (Etxeberria and Sagardui 2008a; Sincero et al. 2010; Siegmund et al. 2011). The real products are first generated and the interdependencies are then measured based on the results of product testing. As it is infeasible to generate the real products for quality testing in practice, the applicability of these approaches is restricted to the domains where the real products can be generated easily, i.e., Linux operating system domain (Sincero et al. 2010). To avoid generating real products, another group of approaches use domain experts’ judgments to measure the interdependencies. The relative impacts of individual features on a quality attribute are assigned with qualitative values (Lee and Kang 2010; Peng et al. 2009; Sinnema et al. 2006; Thurimella et al. 2008) or estimated by quantitative measurement methods (Zhang et al. 2003). However, these approaches can only provide rough quality assessment or require onerous domain experts’ judgments. Two research questions are raised with respect to modeling quality attributes in a feature model.

3. How to better model quality attributes in a feature model.
4. How to enhance the feature-based product configuration process based on the captured quality knowledge.

1.2 Proposed Approach

This thesis proposes a framework that focuses on the quality assessment in software product line development. In this framework, a set of approaches (Figure 1.1) are developed to address the research questions raised in Section 1.1. In Figure 1.1, the arrows represent the production and consumption relations between phases and artifacts and also the order of phases and activities. This framework has three major phases: 1) validate feature models, 2) model quality attributes in a feature model, and 3) quality-aware product configuration. Of these three phases, 1) and 2) are domain engineering activities while 3) is an application engineering activity.
The starting phase is to validate a feature model. The input of this phase is an initial feature model established by feature modeling approaches, i.e., feature oriented domain analysis (FODA) (Kang et al. 1990) and the output of this phase is a valid feature model which contains
no errors. A feature model validation approach is developed to identify and explain feature model errors based on the contradictory feature relationships behind the errors. The second phase aims to model quality attributes in a feature model for product quality assessment. The input of this phase is a valid feature model and the output of this phase is a knowledge database called a quality attribute knowledge base which stores the quality knowledge. A quality attributes modeling approach is developed based on domain experts’ judgments by using analytic hierarchical process (Saaty 2001, 2008), which is a pair-wise comparison method. The final phase is to conduct quality-aware product configuration. The input of this phase is a valid feature model together with the generated quality attribute knowledge base and the output of this phase is a valid product. An approach is developed to support the feature-based product configuration in a quality-aware manner.

1.3 Contributions

The main contributions of this thesis are as follows:

1. Feature Model Validation

- I identify the relations between feature model errors and contradictory relationship sets (CRSs). Based on this identification, a feature model validation approach is proposed to identify feature model errors by finding all CRSs from a feature model. A specific feature model error can be better explained by the CRSs that cause the error. To my knowledge, the current popular validation approaches fully or partially depend on the solvers. The validation approach proposed in this thesis is the first one that works without the need of solvers.
I explore several structural properties of a feature model and show how these properties can be used in the feature relationship propagation, the process of propagating the configuration decision on a specific feature to other features. We further develop an algorithm to implement the feature relationship propagation based on the depth-first search on a feature tree. This propagation algorithm can be easily adapted to support feature-based product configuration in several ways, such as checking the consistency of configuration decisions, calculating the variation degree of feature models (Maben and Lichter 2005) or finding the contradictory feature relationships as used in the developed validation approach.

I empirically show that the current CSP-based validation approaches require a great amount of resources, such as time and memory. The validation approach in this thesis is more efficient than the existing CSP-based approaches for debugging large-scale feature models. We expect that the insights provided could lead to many new efficient algorithms of validating feature models in the future.

2. Quality Assessment

I propose an approach for assessing the quality attributes of a specific product at the stage of feature-based product configuration. The issues that need to be addressed for the quality assessment are explicitly identified, i.e., identifying and representing quality attributes in a feature model, measuring the interdependencies
between features and quality attributes, and conducting quality-aware product configuration. This proposed quality modeling approach can be considered as a starting point of assuring the product quality during the product development in an SPL. It can be followed up by the quality assurance activities in the later stages, i.e., architecture derivation and component composition.

- We use the analytic hierarchical process (AHP) to help domain experts prioritize the relative impacts of individual features on a quality attribute. It is much easier for a domain expert to make the pair-wise comparisons in the AHP method. The consistency of domain experts’ judgments can be ensured by the consistency ratio returned from AHP. We expect that the use of AHP could inspire the future work in two ways: 1) explore the further use of AHP, such as prioritizing the quality attributes of a software product line or finding the best solution for the overall quality; 2) use other techniques or methods for domain experts’ judgments.

- We empirically show that the proposed quality assessment approach can provide the precise assessment to a great extent. 83% of the assessed quality levels match the domain experts’ estimation. Meanwhile, the domain experts’ efforts involved in the judgments are significantly reduced from two aspects: 1) the number of judgments is reduced from $2^n$ (in the worst case) to $n(n-1)/2$; and 2) the time spent for a single judgment is reduced from minutes to seconds on average.
3. Supporting Tools

- A main challenge of evaluating feature model validation approaches is the lack of the large-scale feature models. Currently, the majority of publicly-available feature models are small and not suitable for performance evaluation. Hence, we build a tool that is capable of generating the random feature models based on a set of parameters. The generated models are used in the experiments to support the performance analysis.

- We develop a prototype tool which supports a set of activities from domain engineering to application engineering, such as editing feature models, validating feature models, modeling quality attributes in feature models and configuring products from feature models. The concepts of the proposed framework have been implemented into this prototype tool. The prototype tool has been made public as we hope that other researchers can take advantage of the infrastructure we have built.

We claim that my research work provides a significant step forward for applying feature models in product configuration. Specially, we propose an efficient validation approach for identifying and explaining feature model errors and a systematic quality attributes modeling approach for the quality assessment at the stage of feature-based product configuration. Ultimately, we expect that the ideas proposed in this thesis could inspire other researchers to continue the work in these areas.
1.4 Thesis Structure

This thesis is organized as follows. Chapter 2 introduces the background knowledge that is necessary for reading this thesis, including software product lines, feature models and feature-based product configuration. Two practical problems are identified in the area of feature-based product configuration by reviewing the related work in literature. Chapter 3 proposes a validation approach for identifying and explaining feature model errors based on the contradictory feature relationships behind the errors. The algorithm of finding the contradictory feature relationships is developed based on the feature relationship propagation. Chapter 4 proposes an approach of modeling quality attributes in a feature model based on domain experts’ judgments. The interdependencies between features and quality attributes are explicitly measured by AHP. A quality attribute knowledge base is generated to store the captured quality knowledge which is used for quality-aware product configuration. The prototype tool which implements the concepts of the proposed framework is introduced in Chapter 5. This prototype tool is online available. The results of empirical experiments used for evaluating the proposed approaches are reported in Chapter 6. First, the feature model validation approach is evaluated from the correctness of identified errors and the efficiency of validating feature models. The feature models used in the experiments are presented and the results of validating feature models by the proposed validation approach and CSP-based approaches are presented and discussed. Second, the quality attributes modeling approach is evaluated based on the precision of assessed product qualities. A large case study from the computer aided dispatch software product line is generated. The experimental results of assessing the quality levels for configured products by the proposed quality modeling approach are presented and discussed. Chapter 7
concludes this paper and identifies the future research direction.

1.5 Publication List

The following published papers were based on the work presented in this thesis.


2. Background

This chapter introduces the background knowledge that is necessary for reading this thesis. The study of the background knowledge motivates my interests on two research topics: feature model validation and quality assessment in product configuration. To understand the current research, we review the related works on these two topics and identify the limitations of the current existing works.

2.1 Software Product Lines

Mass production is the process of manufacturing a large number of the same products following a standardized process to enable a reduced time-to-market. Mostly, mass production cannot achieve the customers’ specific requirements for products and mass customization is essential in today’s competitive and segmented market. Mass customization aims to meet as many individual customers’ needs as possible while maintaining the highest mass production efficiency as possible (Benavides et al. 2010). The product line approach was first introduced into manufacturing to achieve mass customization. A product line refers to a number of products that are developed by the same manufacturer and together address a particular market segment or fulfill a particular mission. A product line succeeds because the commonalities shared by the products can be exploited to achieve the economies of production. Many companies and organizations have built their own product lines, such as Airbus, Ford, Dell, Apple and McDonald’s.
The success of product lines in manufacturing inspired the software engineering community.

The software product lines (Clements and Northrop 2002) also called software product families (Pohl et al. 2005) are introduced into software engineering to achieve mass customization of software products. A software product line (SPL) consists of a set of software-intensive systems and these software systems are similar to each other because they are developed to satisfy the specific needs of a particular market or mission (Clements and Northrop 2002). *Software product line engineering* (SPLE) is a new software development paradigm which aims to develop software products by reusing the core assets of a software product line rather than produce them one by one from scratch (Roos-Frantz et al. 2011). SPLE can enable more systematic software reuse than the traditional software reuse. Traditional software reuse is rather ad-hoc and the reusable parts are usually components or codes. Contrarily, the reuse process in SPLE is more strategic and the reusable parts can be derived from any stage of software development life cycles.

There are two distinct development processes in SPLE: development for reuse and development with reuse (Linden et al. 2007). The “development for reuse” process is also called *domain engineering* which aims to define and realize the commonalities and variabilities of the products in a software product line. The “development with reuse” process is also called *application engineering* which aims to derive new software products by reusing the commonalities and resolving the variabilities. The process of SPLE is illustrated in Figure 2.1.
Figure 2.1 Framework of Software Product Line Engineering

Domain engineering is the process of documenting a specific domain and developing a reuse-infrastructure in order to support individual software products development. Domain engineering has three main sub-phases as shown in Figure 2.1: domain analysis, domain design and domain implementation. In domain analysis, the domain engineers define the domain scope by analyzing domain information, such as existing products in the domain, domain experts’ knowledge and experience, handbooks, requirements on future systems, etc. The key issue of domain analysis is to identify the commonalities and variabilities of the products in a software
product line and specify the constraints that restrict the choices on variabilities (Sinnema and Deelstra 2006). The knowledge captured in domain analysis is finally integrated into domain models. In domain design, the software architects develop a reusable architecture for the software product line based on domain models. This reusable architecture is called software product line architecture or family reference architecture (Jazayeri et al. 2000). The software product line architecture is the key artifact in a software product line. In contrast to single software architectures, software product line architectures are designed to underpin multiple systems. They reify the commonalities between the various products and also clearly delineate the variabilities that are allowed between products. Kang et al. (1998) propose the method of developing the reference architectures and software components for reuse based on feature models in feature-oriented reuse method (FORM). Domain design is followed by the domain implementation sub-phase during which the software developers apply proper technologies to implement the reusable components. Although it will certainly require extra effort to develop the reuse-infrastructure that includes domain models, family reference architecture, reusable components and reusable test cases in domain engineering, the effort can be mitigated by the benefits of employing this reuse-infrastructure to develop new software products in application engineering (Deelstra et al. 2005).

Application engineering is the process of deriving new software products from a software product line by reusing as many reusable assets as possible (Pohl et al. 2005). In application engineering, product derivation is the process of selecting the reusable assets and customizing them for a particular product (Rabiser et al. 2010). Application engineering also has three sub-
phases: application analysis, application design and application implementation. In application
analysis, the application engineers elicit, analyze and interpret the customers’ requirements.
Then the product is configured at an abstract level by resolving the variabilities in domain
models based on the customers’ requirements. In application design, the specific product
architecture is derived from the family reference architecture based on the resolved variabilities.
Finally, the generic components are selected and composed into a final product in domain
implementation. In some cases, the customers’ requirements cannot be fully realized by the
current software product line and additional development is needed to satisfy the new
requirements. The domain engineers should consider and decide whether the new design and
implementation should be incorporated into the software product line. In this way, the software
product line will be evolved.

One of the most important issues in software product line engineering (SPLE) is to explicitly
define and manage the variability (Pohl et al. 2005). Herein, the variability is defined as the
ability of a software system or artifact to be extended, changed, customized, or configured for
use in a specific context. Variability management aims to handle the introduction, use and
evolution of the variabilities in the process of SPL (Sinnema et al. 2006). For example, in
domain engineering, the variabilities among the products in a software product line are
identified and realized into various levels of abstraction (Svahnberg 2000). In application
engineering, specific products are derived from a software product line by exploiting the
variabilities along all the phases of software development. Variability management is a very
complicated task, as it deals with a number of complications. First, not all the combinations of
the choices on variabilities are consistent because of the constraints among variabilities, such as choosing a particular artifact may require or exclude the presence of other artifacts. As the number of variabilities and constraints can easily run into hundreds of thousands in an industrial software product line (Sinnema et al. 2006), conflicting constraints may exist which will prevent the effective product derivation. Second, a specific choice on variabilities may influence the levels of a set of quality attributes of the target product simultaneously. Mostly, the precise impact on the quality attributes may not be clear until the final product is produced and tested. However, if it is found that the customers’ quality requirements are not satisfied in the system testing phase, it may be too costly to fix the problems. Therefore, it is essential to pay attention to the quality attributes in variability management which is one of the early development phases in an SPL.

The variability modeling techniques are essential for effective variability management. The main purpose of a variability modeling technique is to represent the variabilities and the constraints among variabilities in a way that provides an increased overview and understanding of the variabilities for the effective product derivation (Sinnema and Deelstra 2006). Many variability modeling techniques have been proposed in literature. For example, one category of approaches manages the variabilities at different abstract levels, such as requirement models, design models and component models, and correlates the variabilities in different models by explicit links (Becker 2003; Pohl et al. 2005; Sinnema et al. 2006). The composition or generalization relationships among different variabilities are not represented in this category. Another category of approaches extends use case models with the mechanisms of representing
variabilities (Clauss 2001a, 2001b; Halmans and Pohl 2003; Maben and Lichter 2002). The relationships among different variabilities are not clearly described in this category, as the variabilities are hidden in the use case descriptions. The third category has different flavors of modeling variability with feature models (Czarnecki and Kim 2005; Kang et al. 1990, 1998; Maben et al. 2004a).

Despite applying different modeling concepts, all the above techniques aim to represent the variabilities of a software product line in a way whereby the variabilities can be managed more easily in domain engineering and can be exploited more easily in application engineering. In the last decade, feature models were most widely used for variability modeling in software product lines. First, a feature model is a simple and comprehensive mechanism for representing the commonalities and variabilities of products in a software product line. Second, a feature model is easy to use and communicate, as different stakeholders usually speak of product characteristics in terms of features the product has or delivers. The next section will introduce the notation of feature models and the product configuration based on feature models.

2.2 Feature Models

A feature model is a formalism of capturing and representing the common and variable characteristics among the products in a software product line (Czarnecki 1998). A feature is defined as “a prominent or distinctive user-visible aspect, quality, or characteristic of a software system or systems” (Kang et al. 1990) or “a system property that is relevant to some stakeholder”
(Czarnecki and Eisenecker 2000). Each valid combination of features authorized by a feature model corresponds to a possible product of the software product line.

The feature model is first introduced into software product lines in feature oriented domain analysis (FODA) (Kang et al. 1990). Since then, FODA has been extended by several works to improve its usefulness. For example, Kang et al. (1998) extend FODA to the software design phase and describe how a feature model can be used to develop reference architectures and software components for reuse. Griss et al. (1998) integrate FODA into reuse-oriented software engineering to support domain engineering and component reuse. Eriksson et al. (2005) use feature models as the main tools for managing variabilities within use case models for the software product line. Although feature models can be used in several aspects of software product line engineering, the most common use is to manage the variabilities in domain engineering and provide configuration rules in application engineering.

2.2.1 Feature Diagram and Product Configuration

A feature model is represented by an AND/OR tree-structure feature diagram which has two kinds of components: features and feature relationships. The features describe distinguishable characteristics of software systems and the feature relationships represent the selection constraints among the features. The feature relationships can be further classified into the hierarchical relationships which describe the selection constraint between a parent feature and its child features, and the cross-tree constraints which represent the selection constraint between two cross-tree features. Each feature diagram has one and only one root feature for representing
the concept of software systems in the domain. Each feature has one and only one parent feature except for the root feature and each feature has a set of child features except for leaf features.

Since Kang et al. (1990) first introduce the notation of basic feature diagram in FODA, several extensions have been proposed to improve its succinctness and naturalness. For example, Riebisch et al. (2002) introduce the unified modeling language (UML) multiplicity to enhance the feature diagram notation. Czarnecki et al. (2004) propose cardinality-based feature model (CBFM) which includes feature cardinalities, group cardinalities and feature diagram references. Benavides et al. (2005) extend feature models with feature attributes to represent quality attributes. Zhang et al. (2004) introduce impact relationships which cannot imply the selection constraints among features, but can represent the impact of the selection or removal of a specific feature on other features. Among the above extensions, all authors agree that a minimum feature diagram notation should be able to present three kinds of hierarchical relationships: “mandatory”, “optional” and “feature groups” and two kinds of cross-tree constraints: “requires” and “excludes” (Mendonca et al. 2008). This thesis adapts the cardinality-based feature model proposed by Czarnecki et al. (2004). The hierarchical relationship between a feature group and its grouped features are considered as the relationship between a variation point and a set of variant features under this variation point. The semantics and notations of feature relationships in the adapted cardinality-based feature model are shown in Table 2.1.
<table>
<thead>
<tr>
<th>Feature Relationships</th>
<th>Semantic</th>
<th>Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hierarchical</td>
<td>Mandatory</td>
<td>if the parent feature is selected, the child feature which has a mandatory relationship with its parent must be selected.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td>if the parent feature is selected, the child feature which has an optional relationship with its parent may or may not be selected.</td>
</tr>
<tr>
<td>Variation</td>
<td></td>
<td>if a variation point (VP) is selected, at least ( a ) variant features must be selected and at most ( b ) variant features can be selected from the group of variant features based on the cardinality ([a...b]).</td>
</tr>
<tr>
<td>Point and Variant</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cross-Tree Constraints</td>
<td>Requires</td>
<td>“A requires B” means that if feature A is selected, feature B must be selected.</td>
</tr>
<tr>
<td></td>
<td>Excludes</td>
<td>“A excludes B” means that A and B cannot be selected into the same member product.</td>
</tr>
</tbody>
</table>
Figure 2.2 shows an example of the cardinality based feature model from the tourist guide software product line. *Tourist Guide* is the root feature. If *Service* is selected, *Route Search* must be selected because of the mandatory relationship between them and *Position Detection* may or may not be selected because of the optional relationship between them. If *Terminal Device* is selected, at least one from *Mobile* and *PDA* must be selected due to the group cardinality “1…2”. The feature model of Figure 2.2 is used as an example to illustrate the approaches proposed in this thesis.

![Figure 2.2 A Feature Model of Tourist Guide System Software Product Line](image)

To maintain the information of the cross-tree constraints, a matrix-based approach proposed in (Ye and Liu 2005) is used. First, a matrix with *n* columns and *n* rows is generated where *n* is the total number of features involved in the cross-tree constraints. Second, all the features involved...
in the cross-tree constraints are listed as the column elements and row elements respectively in the generated matrix. Finally, if a feature in the row requires a feature in the column, the corresponding cell is assigned with “1”, while if a feature in the row excludes with a feature in the column, the corresponding cell is assigned with “2”. Following the above steps, a matrix as shown in Table 2.2 is generated to manage the constraints of the feature model of Figure 2.2. From this matrix, it can be found that Mobile excludes Encryption, Terminal Device excludes Modem, and Terminal Device requires Encryption etc.

Table 2.2 Matrix for Maintaining Constraints in Tourist Guide Software Product Line

<table>
<thead>
<tr>
<th></th>
<th>Mobile</th>
<th>WAN</th>
<th>LAN</th>
<th>Modem</th>
<th>Web-Based</th>
<th>Terminal Device</th>
<th>Encryption</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mobile</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WAN</td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LAN</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Modem</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Web-Based</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Terminal Device</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Encryption</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In a feature model, the features can be classified into full-mandatory features and variable features. A full-mandatory feature is a feature which connects with a root feature by a hierarchical path where all hierarchical relationships are mandatory (Maben and Lichter 2004). Based on the semantics of mandatory relationships, it can be known that a full-mandatory feature will appear in all products in a software product line. A feature is a variable if it is not a full-mandatory feature and a variable feature may or may not be present in a product. The
“requires” or “excludes” constraints can only exist between two variable features (Ye et al. 2010). In the feature model of Figure 2.2, the full-mandatory features include \textit{Tourist Guide}, \textit{Security Mechanism}, \textit{Operating Environment}, \textit{Network Connection}, \textit{Services} and \textit{Route Search}, and all other features are variable features.

The product configuration based on a feature model is the process of selecting the desired features based on the customers’ product requirements, but the selections must satisfy the constraints specified in the feature model. As features are the units for selection, the product configuration based on a feature model is also called feature-based product configuration (FBPC). In FBPC, different stakeholders make the feature selections from different perspectives. For example, the sales people and project managers make the selection of functional features based on the customers’ functional needs while the software developers and software architects make the selection of technical features (i.e., design decisions and implementation techniques) based on the customers’ quality requirements. In practice, FBPC is rarely a sequential process and several iterations are necessary for eliciting the customers’ requirements and resolving the variabilities (Rabiser et al. 2010). A product derived from FBPC is actually a collection of legally selected features. From the feature model of Figure 2.2, a specific product can be derived and consists of the following features: \textit{Tourist Guide}, \textit{Security Mechanism}, \textit{Operating Environment}, \textit{Network Connection}, \textit{WAN}, \textit{Terminal Device}, \textit{PDA}, \textit{Encryption}, \textit{Position Detection}, \textit{Satellite}, and \textit{Web-Based}. 

27
2.2.2 Two Problems

Although the feature models are widely used for product configuration in software product lines, not all related problems are addressed properly by the current research. Two problems arise as the major obstacles of applying feature models in product configuration in practice. Firstly, the model designers will inevitably introduce some contradictory feature relationships into feature models by mistake. The contradictory feature relationships are a set of feature relationships that will result in contradictions in product configuration. For instance, Mobile in Figure 2.2 is a variable feature which means that it can be either selected or removed in product configuration. When Mobile is selected, its parent feature Terminal Device must also be selected. In order to satisfy the constraint “Terminal Device requires Encryption”, Encryption must be selected due to the inclusion of Terminal Device. If Encryption is selected, feature Mobile must be removed to satisfy the constraint “Encryption excludes Mobile”, which is obviously a contradiction since feature Mobile has already been selected. These contradictory feature relationships will ultimately result in feature model errors, i.e., dead features and false variable features (Benavides et al. 2010). In the above example, the contradictory feature relationships lead to an error on Mobile, because Mobile cannot be selected in product configuration. To ensure the effective product configuration, the errors existing in a feature model must be identified and corrected in feature model validation. An automated validation approach is very important, as validation of large-scale feature models is time-consuming and error-prone.
Another problem concerns the product quality assessment at the stage of feature-based product configuration (FBPC). In application engineering, the functional needs of the target product can be determined by selecting the desired features in FBPC. However, only the functionalities are not sufficient for the satisfaction of the final product. The product quality is also the major concerns of different stakeholders, especially in some specific domains, such as safety in the safety-critical domain, performance in the real-time domain and reliability in the embedded system domain. As it is known, the product quality can be affected by all the life cycles of product development, such as product configuration, software architecture derivation and reusable components composition. Mostly, quality is not considered until the final product is generated and tested in the system testing phase (Montagud and Abrahao 2009). However, if it is found that the product’s quality cannot satisfy the customers’ quality requirements at the latter stages of product development; it will be very costly to fix the problems as the customers would need to return to the beginning of the development life cycles to re-configure the product. Therefore the estimation and prediction on the product quality should be considered as early as possible in application engineering. As FBPC is considered as the first stage of prevention of deriving invalid products in product development, the product quality should be assessed at this stage. To understand the current research on the above two problems, we will review the related works on feature model validation and quality assessment in product configuration.
2.3 Feature Model Validation

The feature model validation aims to identify the errors existing in a feature model and provide explanations for each identified error (Batory et al. 2006). This section will introduce feature model errors and review the related works of feature model validation.

2.3.1 Feature Model Errors

Trinidad et al. (2008) considers a feature model error as an incorrect definition of feature relationships which may lead to the set of products described by a feature model not matching the software product line it describes. In literature, three critical feature model errors have been proposed: void feature model (Benavides et al. 2005), dead feature (Trinidad et al. 2008), and false variable feature (Trinidad et al. 2008). Benavides et al. (2010) have summarized the definitions of these three feature model errors as follows:

- **Dead features**: A feature is dead if it cannot appear in any product line member despite being defined in a feature model.

- **False variable features**: A feature is false variable if it must be present in a product whenever its parent is present, despite not being modeled as a mandatory feature.

- **Void feature model**: A feature model is invalid or void when no products can be configured from the feature model. A void feature model is a special case of dead feature error, where all features in the feature model are dead.
The feature model of Figure 2.3 is used to illustrate the feature model errors. In Figure 2.3, \( R \) is the root feature and it will be included in all products; feature \( A, B, C \) and \( D \) connects with \( R \) by the hierarchical relationships \( Br-1, Br-2, Br-3 \) and \( Br-4 \) respectively; feature \( E \) and \( F \) connects with feature \( A \) by optional relationship \( Br-5 \) and \( Br-6 \); feature \( G \) and \( H \) constitute a feature group with cardinality “1..1” and connect with their parent feature \( D \) by \( Br-7 \); feature \( E \) excludes with feature \( B \) by the “excludes” constraint \( Ex-1 \); feature \( C \) excludes with feature \( D \) by the “excludes” constraint \( Ex-2 \); feature \( B \) requires feature \( F \) by “requires” constraint \( Rq-1 \); feature \( G \) requires feature \( C \) by “requires” constraint \( Rq-2 \).

![Figure 2.3 Examples of Feature Model Errors](image)

Based on the above definitions of feature model errors, a set of errors can be identified in the feature model of Figure 2.3. Feature \( G \) and \( E \) are dead features because they cannot appear in any product. Feature \( H \) and \( F \) are false variable features because they must be present in all products. The causes of these errors can be found by examining the feature relationships. As Ye
et al. (2010) discuss, the inclusion of full-mandatory features in “requires” or “excludes” constraints will result in feature model errors, as the full-mandatory features are not selectable. Therefore, $E$ is a dead feature as it excludes with the full-mandatory feature $B$ and $F$ is a false variable feature as it is required by the full-mandatory feature $B$. The errors on $G$ and $H$ are caused by the contradictory feature relationships. $G$ is a dead feature due to the set of contradictory feature relationships: $Rq\cdot2$, $Ex\cdot2$ and $Br\cdot7$, while $H$ is a false variable feature due to the set of contradictory feature relationships $Br\cdot7$, $Rq\cdot2$ and $Ex\cdot2$. The feature model of Figure 2.3 is not a void feature model because at least one product which consists of $R$, $B$ and $F$ can be configured from the feature model.

As discussed earlier, two important tasks need to be accomplished in feature model validation: error identification and error explanation.

- **Error Identification**: In the task of error identification, a feature model is input and a list of feature model errors is returned, i.e., dead features, false variable features and void feature model. For instance, if the feature model of Figure 2.3 is input for error identification, four errors will be returned: dead feature $G$, dead feature $E$, false variable feature $H$ and false variable feature $F$.

- **Error Explanation**: In the task of error explanation, a feature model error is input and a set of its explanations is returned. Herein, an explanation of a feature model error is a right set of feature relationships that must be modified to remove the error.
(Trinidad et al. 2008). Each feature model error may correspond to a set of explanations. For instance, if “dead feature $G$” is input for error explanation, three explanations will be returned: $\{Br-2\}$, $\{Ex-2\}$ or $\{Br-7\}$.

### 2.3.2 Related Works

The feature model validation is first identified in FODA (Kang et al. 1990) as a fundamental task for feature modeling in software product lines. Since then a set of validation approaches has been proposed. For example, Zhang et al. (2004) transform a feature model into propositional formulas and use an SVM system to detect dead features and full-mandatory features. Czarnecki and Kim (2005) transform a feature model into a constraint satisfaction problem (CSP) and use CSP solvers to identify void feature models and dead features. Trinidad et al. (2008) also transform a feature model into a constraint satisfaction problem and use the CSP solvers to detect the explanations of a feature model error.

The above approaches formalize a validation problem into a constraint satisfaction problem (CSP) or Boolean satisfaction problem (SAT) which is a certain form of CSP where the domains of variables are Booleans. Then the off-the-shelf tools (i.e. SAT-solvers, CSP-solvers, and BDD-solvers) are used to automate the feature model validation. However, the CSP is NP-complete in theory (Cook 1971) and it exhibits exponential complexity (Mackworth 1977; Segura 2008; Yan et al. 2009), requiring a combination of heuristics and combinational search methods. Due to the NP-complete nature, the validation of a large-scale feature model in the above approaches, especially for explaining feature model errors, suffers from state space
explosion problem. For example, Trinidad et al. (2008) transform a feature model into a constraint satisfaction problem where all features are transformed into feature variables and all feature relationships are transformed into constraint variables. Then the feature model errors and their explanations are identified based on whether valid solutions can be derived from the constraint satisfaction problem under the certain conditions. Using Trinidad’s approach, to validate a feature model with $n$ features and $m$ feature relationships, the state space of the constraint satisfaction problem is as large as $2^{m+n}$ (Trinidad et al. 2008). In this case, it is impossible to explore the entire state space with limited resources of time and memory by the general CSP solvers.

To improve the validation efficiency, a set of approaches aims to reduce the size of the CSP that needs to be resolved by the solvers without changing the result of feature model validation. For example, Yan et al. (2009) propose an approach of eliminating the validation-irrelevant features and feature relationships from a feature model. Segura (2008) proposes an approach of finding atomic sets, the group of features that can be treated as a unit in product configuration and using atomic sets instead of features in feature model validation. Mendonca (2009) develops a propagation algorithm to deal with the hierarchical relationships and uses the SAT-solvers to reason on the extra-constraints, such as requires and excludes. As discussed, the complexity of resolving the CSP of a feature model using solvers is exponential to the number of units (features and relationships) included in the feature model. In this sense, the above approaches can improve the validation efficiency in some domain specific cases, such as if the number of
validation-irrelevant units is large, as in Yan’s work, the number of mandatory relationships is large, as in Segura’s work, or the number of extra-constraints is small, as in Mendonca’s work.

In conclusion, the current validation approaches transform a feature model into a constraint satisfaction problem (CSP) and then use the off-the-shelf solvers to reason on the CSP (Czarnecki and Kim 2005; Trinidad et al. 2008; Zhang et al. 2004). These approaches are quite inefficient, as they need to use the solvers to explore the entire solution space of the CSP for finding the explanations of feature model errors. A set of approaches is proposed to reduce the size of the CSP that needs to be resolved by the solvers without changing the validation results in different ways (Mendonca 2009; Segura 2008; Yan et al. 2009). Although these approaches can improve the validation efficiency in some domain specific cases, the complexity is still exponential to the number of the units in a feature model in the general cases. A more efficient feature model validation approach is needed.

### 2.4 Quality Assessment in Product Configuration

As discussed in Section 2.2.2, the product quality should be assessed at the stage of product configuration from feature models. To achieve this aim, the quality attributes need to be modelled in a feature model. This section will introduce the issues that need to be addressed for the quality attributes modeling and review the current related works.

#### 2.4.1 Quality Attributes Modeling in Feature Models

Modeling quality attributes in a feature model is still a challenging research direction in the software product line community, as the quality attributes always have vague definitions and no
clear-cut satisfaction criterion. As it is known, the key issue of modeling quality attributes in a feature model is to measure the interdependencies between the features and quality attributes, because different feature selections have a different impact on the quality level of a configured product. Two important issues need to be addressed for measuring the interdependencies. First, it is necessary to estimate what influence the inclusion of an individual feature will have on a quality attribute, as individual features will have a different impact on a quality attribute. Second, it is necessary to identify how to aggregate the influence of a set of features included in a configured product, as features may affect a quality attribute interdependently.

Once the interdependencies between features and quality attributes have been measured, the feature-based product configuration can be enhanced in a quality-aware manner. The quality-aware product configuration (QAPC) is the process of deriving a product from a feature model with a full consideration of the customers’ functional needs and quality requirements. In QAPC, to satisfy one quality attribute, the customers may have to sacrifice some other quality attributes. For example, performance and security often exist in a state of mutual tension. A secure system often has the mechanisms to prevent attacks from security breaches. However, the inclusion of the security mechanisms into the final product will affect performance negatively, as respond time is extended. Thus the relationships among different quality attributes need to be recognized for QAPC.
2.4.2 Related Works

In literature, a set of approaches have been proposed to model quality attributes in feature models since first suggested in feature oriented domain analysis (FODA) (Kang et al. 1990). Among these approaches, the most important ones have been chosen for review. The reviewed approaches can be classified into three categories based on the method used for measuring the interdependencies between features and quality attributes:

- real products based
- domain experts’ judgments based (qualitative)
- domain experts’ judgments based (quantitative)

The approaches in the first category measure the interdependencies between features and quality attributes based on the real products. For example, Etxeberria and Sagardui (2008a, 2008b) extend a feature model with a utility tree to characterize quality attributes. The impacts of individual features on a quality attribute is measured by evaluating software architectures derived via some generic architecture evaluation methods or the execution of final produced products. In the case that a generic evaluation model cannot be constructed due to the nature of the quality attribute, the evaluation must be performed via the execution of the produced products. Siegmund et al. (2010, 2011) develop an integrated software product line model that includes a feature model and an implementation model. The feature model consists of high level domain variation points (i.e., services and functions) and the implementation model consists of low level implementation variation points (i.e., design and implementation). To approximate the
impact of a specific feature on a quality attribute, a set of products that differs in the single feature are produced and the difference values on the quality attribute in the generated products are interpreted as the approximation of the feature. The interactions among different features with respect to affecting a quality attribute can be identified from the measurement results.

Sincero et al. (2007, 2010) propose a feedback approach to achieve a higher degree of configurability by approaching the non-functional properties (NFPs) in the context of software product lines. This approach collects the information about NFPs by generating and testing the real products in compile-time or runtime, processing the raw results and storing the influence of each feature and the combination of features on a NFP into a NFP database. The approaches in this category measure the interdependencies by evaluating real products. The relative impacts of individual features on a quality attribute are evaluated numerically with continuous values and the quality level of a configured product can be calculated based on a set of quantitative values. However, it is not feasible to generate real products in practice and it is costly and time-consuming to generate a large number of real products for quality evaluation. Although some approaches aim to reduce the measurement costs by selecting a small but suitable set of products for evaluation by analyzing feature relationships (Etzeberria and Sagardui 2008b, Sincero et al. 2010), requiring the real products is still the main limitation of the approaches in this category.

To avoid generating the real products, a set of approaches uses domain experts’ judgments to measure the interdependencies. These approaches can be further classified into qualitative method based and quantitative method based. A qualitative method concentrates on using some
qualitative values to indicate the relative impacts while a quantitative method is concerned with estimating the relative impacts numerically with continuous values. The approaches in the second category use the qualitative methods for measuring the interdependencies. For example, Lee and Kang (2010) develop a coherent domain knowledge model to represent the variability on usage contexts, quality attributes and product functionalities and the relationships between quality attributes and features are represented as qualitative values. Thurimella et al. (2008) extend the feature model by augmenting selection criteria, such as usability and availability, and assessing each variable feature based on its related criteria in qualitative values. Sinnema et al. (2006) use a dependency to specify a system property and how the selection of variant features influences the value of the system property. Peng et al. (2009) propose a context-based method for the non-functional variability analysis and extend a feature model with non-functional goals and non-functional related constraints based on a non-functional requirement (NFR) framework. Jarzabek et al. (2006) develop a feature-softgoal interdependency graph which records design rationale by the explicit and implicit contributions from features to quality attributes. The approaches in this category use the qualitative methods to measure interdependencies based on domain experts’ judgments. The relative impacts of features on a quality attribute are some qualitative values assigned directly by domain experts. The quality of a configured product is estimated by combining a set of qualitative values. Therefore, these approaches lack sufficient quantitative data to actually determine the best satisfying product for the customers and they only support quality assessment in a rough manner.
To overcome the limitations of the above approaches, the third category uses quantitative methods to measure the interdependencies based on domain experts’ judgments. For example, Zhang et al. (2003) propose an approach of modeling the impact of system variants on quality attributes based on Bayesians belief network (BBN) and use the BBN model to predict and assess the quality attributes of a configured product. A BBN model includes a set of nodes for representing design decisions and quality attributes and directed edges for representing the influential relationship between two nodes. A probability number is assigned to reflect domain experts’ belief in how much a given partial configuration influences a quality attribute, such as

\[ P(QA = \text{high} \mid VarA, VarB) = p. \]

This formula means that when variable feature \( A \) and variable feature \( B \) are included in a configured product, the probability that \( QA \) of the configured product is high is \( p \). These probability values are assigned by domain experts and are based on their knowledge and experience accumulated from similar projects in the domain. In product configuration, the quality level of a configured product can be predicted by quantitative analysis over the BBN model. The quantitative based approach can support the quality assessment in a relatively precise manner. Although the valid products are not required, the human efforts involved in the judgments are very onerous. For example, domain experts need to provide judgments for any combinations of relevant features of a quality attribute in the BBN approach. In the worst case where no feature relationships exist among \( n \) relevant features of a quality attribute, the number of judgments that need to be made is \( 2^n \). It is also difficult for a domain expert to provide judgments for a configuration which has a large number of features.
In conclusion, the above three categories of approaches have advantages and limitations. The first category supports the quality assessment in a precise manner, as the interdependencies are measured based on real products. However, it is difficult and time-consuming to produce real products in practice. The second category avoids generating real products by using domain experts’ judgments. However, this category can only provide rough quality assessment, as the relative impacts of individual features on quality attributes are represented as qualitative values.

To overcome the limitations of the first two categories, the third category uses quantitative methods to measure the interdependencies based on domain experts’ judgments. However, the efforts involved in the judgments are very onerous and it is quite difficult for a domain expert to make the judgments for a combination of features. A quality attributes modeling approach is needed to reduce domain experts’ efforts.

### 2.5 Summary

This chapter first introduces the background knowledge, including software product lines, variability management, feature models and feature-based product configuration. Two problems are identified as the main obstacles of using feature-based product configuration in practice. First, the errors existing in a feature model will prevent the effective product configuration. The feature model errors must be identified and corrected in feature model validation. The current validation approaches are quite inefficient when dealing with large-scale feature models, as they all use solvers to explore the entire solution space of a constraint satisfaction problem which is NP-complete. A more efficient validation approach that can work without the solvers needs to be developed in this research. Second, the quality levels of a target product should be assessed
in feature-based product configuration, as it is very costly to assess the product quality at the later stages of product development. To achieve quality assessment for feature-based product configuration, the quality attributes need to be modelled in feature models. The current modeling approaches either require the real products which are difficult to obtain in practice or involve onerous domain experts’ judgments. An approach of modeling quality attributes in a feature model based on domain experts with reduced efforts for judgments needs to be developed.
3. An Efficient Approach for Feature Model Validation

This chapter proposes an efficient feature model validation approach based on the structural properties of a feature model. As discussed in Section 2.3.1, the feature model errors are caused by the contradictory feature relationships in a feature model. Based on this observation, feature model errors are identified and explained based on the contradictory feature relationships behind the errors. An algorithm is developed to find the contradictory feature relationships from a feature model by identifying the contradictions in feature relationship propagation.

3.1 Causes of Feature Model Errors

This section describes the causes behind feature model errors. As Ye et al. (2010) discuss, the full-mandatory features involved in the cross-tree constraints will result in feature model errors. For example, if a variable feature excludes with a full-mandatory feature, it will be a dead feature as it cannot appear in any product, such as feature E in Figure 3.1. If a variable feature is required by a full-mandatory feature, it will be a false variable feature as it will appear in all products, such as feature F in Figure 3.1. The feature model errors in this case can be resolved by either removing the cross-tree constraints imposed on full-mandatory features or by modifying the hierarchical relationships to change full-mandatory features into variable features. Once all feature model errors caused by the cross-tree constraints imposed on full-mandatory features are resolved, the feature model is not void as at least one product which consists of all the full-mandatory features can be derived from the feature model.
Figure 3.1 Examples of Feature Model Errors (same with Figure 2.3)

Besides the errors in the above case, it is observed that feature model errors (i.e., dead feature or false variable feature) are caused by contradictory feature relationships. The dead feature is used as an example to illustrate this idea. A dead feature will not be present in any product, which means it cannot be selected in product configuration. This is because the selection of a dead feature will result in contradictory configuration conclusions through feature relationship propagation. Herein, the feature relationship propagation starting from a specific feature $f$ is the process of propagating the inclusion or removal of $f$ to a set of other features by the transmissible feature relationships. A contradictory configuration conclusion is a conflict in feature relationship propagation and arises in two scenarios: a feature that has already been included needs to be removed or a feature that has already been removed needs to be included. For a specific feature $f$, if its inclusion or removal results in contradictory configuration conclusions through a set of feature relationships, it can be said that there is an error on feature $f$ and this set of feature relationships is one cause of the error. This research focuses on
resolving the feature model errors caused by contradictory feature relationships. Then the dead features and false variable features can be defined based on their causes as follows:

- **Dead feature**: A feature \( f \) is a dead feature if the inclusion of \( f \) will result in contradictory configuration conclusions by feature relationship propagation starting from \( f \).

- **False variable feature**: A feature \( f \) is a false variable feature if the removal of \( f \) will result in contradictory configuration conclusions by feature relationship propagation starting from \( f \).

Thus, the key issue of identifying an error on feature \( f \) is to find the contradictory configuration conclusions through the feature relationship propagation starting from \( f \). The key issue of explaining an error on \( f \) is to find the cause of the error, which is the set of feature relationships that will result in the contradictory configuration conclusions. In this research, a set of feature relationships that will result in a contradictory configuration conclusion is called a *contradictory relationship set* (CRS). The CRS \((f, \text{inclusion})\) is used to represent a CRS by which the inclusion of \( f \) can result in a contradictory configuration conclusion and CRS \((f, \text{removal})\) is used to represent a CRS by which the removal of \( f \) can result in a contradictory configuration conclusion. In some cases, the inclusion or removal of a specific feature \( f \) will result in more than one contradictory configuration conclusion. As each contradictory configuration conclusion is caused by a specific CRS, the corresponding error on \( f \) is caused by all its related
CRSs. For example, in Figure 3.1, the feature $G$ is a dead feature and its CRS is $CRS(G, \text{inclusion}) = \{Rq-2, Ex-2, Br-7\}$.

Based on the causes of feature model errors, the formal rules of identifying feature model errors are proposed as follows:

- If there exists one or more $CRS(f, \text{inclusion})$ in a feature model, feature $f$ is a dead feature.

- If there exists one or more $CRS(f, \text{removal})$ in a feature model, feature $f$ is a false variable feature.

In order to identify feature model errors and find their explanations, all contradictory relationship sets existing in a feature model need to be found. As a CRS is found whenever a contradictory configuration conclusion arises in feature relationship propagation, the next section will introduce the process of feature relationship propagation and the detailed propagation rules.

### 3.2 Feature Relationship Propagation

The feature relationship propagation is the process of propagating the inclusion or removal of a specific feature to other features in the feature model through different feature relationships, such as hierarchical relationships and cross-tree constraints. In feature relationship propagation
starting from a specific feature $f$, the inclusion or removal of $f$ will lead to its adjacent features, such as parent, child and dependent features, being included or removed based on their specific relationship with $f$. Then the inclusion or exclusion of its adjacent features will result in inclusions or exclusions of their adjacent features as well, and so forth. Then it can be found that the key issue of the feature relationship propagation is the rules about how to propagate the inclusion or removal of a specific feature to its adjacent features. To propose the propagation rules, the following definitions are first provided.

**Definition 1:** Each feature in a feature model has an attribute named as “status” that ranges over the domain {$-1, 0, 1$}. The default “status” value of a feature is “0” which means a decision has not been made on this feature. In product configuration, the attribute “status” gets the value “1” if this feature is included to be a part of the product and gets the value “-1” if this feature is removed from the product.

**Definition 2:** In product configuration, the action of changing the “status” value of a feature $f$ from “0” to “1” is named as including $f$ and represents as $Include (f)$ while the action of changing the “status” value of a feature $f$ from “0” to “-1” is named as removing $f$ and represented as $Remove (f)$.

**Definition 3:** In a feature model, a feature group $fg$ can be represented as $\{f_1, f_2, \ldots, f_n\}$. The number of included features in $fg$ is represented as $NumberOfIncluded (fg)$. The number of
removed features in \( fg \) is represented as \( \text{NumberofRemoved}(fg) \). The total number of features in \( fg \) is represented as \( \text{NumberofFeatures}(fg) \).

**Definition 4:** The adjacent features of a feature \( f \) include all the features that connect with \( f \) by a single feature relationship, such as mandatory, optional, feature group, requires or excludes. The adjacent features of feature \( f \) are represented as follows:

- **Parent feature:** \( f \) has a parent feature if \( f \) is not root feature of a feature model. \( f.\ parent \) is used to represent the parent feature of \( f \). If \( f \) is a root feature, \( f.\ parent \) has the value “null”. \( f.\ pr \) is used to represent the hierarchical relationship that connects feature \( f \) with its parent feature.

  - \( f.\ pr = m \), if a mandatory relationship connects \( f \) with \( f.\ parent \).
  - \( f.\ pr = o \), if an optional relationship connects \( f \) with \( f.\ parent \).
  - \( f.\ pr = v \), if a variation point relationship connects \( f \) with \( f.\ parent \).

- **Child feature:** \( f \) has a number of child features if \( f \) is not leaf feature of a feature model. \( f.\ child \) is used to represent the set of child features of feature \( f \). If \( f \) is not a variation point feature, \( f.\ child \) includes two subsets \( f.\ child-mandatory \) and \( f.\ child-optimal \) while if \( f \) is a variation point feature, \( f.\ child \) includes one subset \( f.\ child-variants \).
The notation \( f. \text{child-mandatory} \) represents the set of child features which connect with \( f \) by mandatory relationships.

The notation \( f. \text{child-optional} \) represents the set of child features which connect with \( f \) by optional relationships.

The notation \( f. \text{child-variants} \) represent the set of variant features that connect with \( f \) by variation point cardinality.

- **Brother feature**: \( f \) has a set of brother features if \( f \) is a variant feature under a variation point. \( f. \text{brother} \) is used to represent the set of variant features which belong to the same variation point with \( f \).

- **Friend feature**: \( f \) has a set of friend features if \( f \) connects with other features by the cross-tree constraints. \( f. \text{friend} \) is used to represent the set of features which connect with \( f \) by “requires” or “excludes”. The feature set \( f. \text{friend} \) can be decomposed into three subsets \( f. \text{exld} \), \( f. \text{reqed} \) and \( f. \text{reqs} \) where \( f. \text{friend} = f. \text{exld} \cup f. \text{reqed} \cup f. \text{reqs} \).

  - The notation \( f. \text{exld} \) represents the set of features that \( f \) excludes.
  
  - The notation \( f. \text{reqed} \) represents the set of features that require \( f \).
  
  - The notation \( f. \text{reqs} \) represents the set of features that \( f \) requires.

Based on the above definitions, the adjacent features for any specific feature in a feature model can be represented. The tourist guide SPL feature model of Figure 2.2 is used to show how to
represent the adjacent features of a specific feature in a feature model. The adjacent features of two features *Terminal Device* and *LAN* are given in Table 3.1

| Table 3.1 Illustration of Adjacent Features in Tourist Guide Software Product Line |
|--------------------------------------|----------------------|----------------------|
| *Feature* f                         | *Terminal Device*    | *LAN*                |
| *f. parent*                          | {Operating Environment} | {Network Connection} |
| *f. child-mandatory*                 | {∅}                  | {∅}                  |
| *f. child-optional*                  | {∅}                  | {∅}                  |
| *f. child-variants*                  | {Mobile, PDA}        | {∅}                  |
| *f. brother*                         | {∅}                  | {WAN}                |
| *f. exld*                            | {Modem}              | {∅}                  |
| *f. reqed*                           | {WAN}                | {Web-based}          |
| *f. reqs*                            | {Encryption}         | {Terminal Device, Web-based} |

In order to simplify the feature model, the optional relationships are first transferred into variation point relationship. In a cardinality-based feature model, a feature *f* and its optional child features *f. child-optional* can be transformed into a variation point with cardinality [0…n] where *n* is the number of optional child features. If there is only one feature in a variation point (*n*=1), the cardinality of the variation point has “1” as an upper bound. For instance, Figure 3.2 shows the tourist guide SPL feature model after transforming all optional features into variation points. The *VirusFilter* and *Encryption* are two optional features under *Security Mechanism* in Figure 2.2 and they are transformed into a variation point with cardinality [0…2] in Figure 3.2. The *Satellite* is an optional feature under *Network Connection* in Figure 2.2 and it is transferred into a variation point with cardinality [0…1] in Figure 3.2. It can be said that feature model of Figure 2.2 is the same as the feature model of Figure 3.2, as they indicate the same set of products. After the above transformation, only four kinds of feature relationships need to be
considered in a feature relationship propagation process: requires, excludes, mandatory and variation point with cardinality.

In feature relationship propagation, the inclusion or removal of a specific feature has different impacts on its adjacent features based on the semantics of different feature relationships specified in Table 2.1. The detailed rules of feature relationship propagation are defined as follows. In these propagation rules, Include \((f)\) is true if \(f\) is included into a product while Remove \((f)\) is true if \(f\) is removed from a product. The propositional logic “A=>B” means if A is true, B will be true.

1. **When the action imposed on \(f\) is Include \((f)\) which means \(f\) is included into a product, the adjacent features of \(f\) will be affected through feature relationships as follows:**
• **Parent feature:** when $f$ is not the root feature, if $f$ is included, its parent feature will be included. The propagation rule is:

\[ \text{Include} (f) \land (f. \text{parent} \neq \text{null}) \Rightarrow \text{Include} (f. \text{parent}) \]

• **Mandatory child feature:** when $f$ is not leaf feature and $f$ has a set of mandatory child features $f.\text{child-mandatory}$, if $f$ is included, all features in $f.\text{child-mandatory}$ will be included. It should be noted that a variation point feature is the mandatory child feature of its parent feature.

\[ \text{Include} (f) \Rightarrow \forall fcm \in f.\text{child-mandatory} : \text{Include} (fcm) \]

• **Variant child feature:** if $f$ is a variation point feature and it has a group of variant child features $f.\text{child-variants}$ with group cardinality of $[a...b]$, when $f$ is included and the number of removed features in $f.\text{child-variants}$ reaches the maximum allowed value “$\text{Number of Features} (f.\text{child-variants})-a$”, all features that are not determined in $f.\text{child-variants}$ will be included. The propagation rule is:

\[ \text{Include} (f) \land \text{Number of Features} (f.\text{child-variants}) - a = \text{Number of Removed} (f.\text{child-variants}) \Rightarrow \forall fcv \in f.\text{child-variants} \land fcv. \text{status} = 0 : \text{Include} (fcv) \]
• **Brother feature:** when $f$ is a variant feature under a variation point feature with cardinality $a...b$ and it has a set of brother features $f.brother$, if $f$ is included and the number of included features in $f.brother$ reaches its maximum value “$b-1$”, all the features in $f.brother$ that are not determined will be removed. The propagation rule is:

\[ \text{Include } (f) \land \text{NumberofIncluded } (f.brother) = b-1 \Rightarrow \forall fb \in f.brother \land fb. \text{status} = 0 : \text{Remove } (fb) \]

• **Requires friend feature:** when there exists a set of features that $f$ requires, if $f$ is included, all features in $f.\text{reqs}$ will be included. The propagation rule is:

\[ \text{Include } (f) \Rightarrow \forall fr \in f.\text{reqs} : \text{Include } (fr) \]

• **Excludes friend feature:** when there exists a set of features that $f$ excludes, if $f$ is included, and then all the features in $f.\text{exld}$ will be removed. The propagation rule is:

\[ \text{Include } (f) \Rightarrow \forall fr \in f.\text{exld} : \text{Remove } (fr) \]

2. **When the action on $f$ is Remove ($f$) which means $f$ is removed from products, the adjacent features of $f$ will be affected through feature relationships as follows:**

- The adjacent features of $f$ will be removed.
• **Mandatory parent feature:** when \( f \) is not the root feature and \( f \) connects with its parent feature by mandatory relationship, if \( f \) is removed, its parent feature \( f. parent \) will be removed. The propagation rule is:

\[
\checkmark \quad \text{Remove} (f) \land f. pr = = m \Rightarrow \text{Remove} (f. parent)
\]

• **Variation point parent feature:** when \( f \) is a variant feature under a variation point feature with cardinality \([a...b]\) and it has a set of brother features \( f. brother \), if \( f \) is removed and the number of removed features in \( f.brother \) exceeds its maximum value “\( \text{NumberOfFeatures} (f.brother)-a' \)”, its parent feature \( f. parent \) will be removed. The propagation rule is:

\[
\checkmark \quad \text{Remove} (f) \land \text{NumberOfRemoved} (f.brother) > \text{NumberOfFeatures} (f.brother) – a \Rightarrow \\
\quad \text{Remove} (f. parent)
\]

• **Child feature:** when \( f \) is not a leaf feature and it has a set of child features \( f. child \), if \( f \) is removed, all the features in \( f. child \) will be removed. The propagation rule is:

\[
\checkmark \quad \text{Remove} (f) \Rightarrow \forall fc \in f.child : \text{Remove} (fc)
\]

• **Brother feature:** when \( f \) is a variant feature under a variation point with cardinality \( a...b \) and it has a set of brother features \( f.brother \), if its parent feature \( f. parent \) has already been included and the number of removed features in \( f.brother \) reaches its maximum value
“NumberOfFeatures (f.brother)-a”, all the features that are not determined in f.brother will be included. The propagation rule is:

✓ Remove (f) \( \text{NumberOfRemoved (f.brother)} = = \text{NumberOfFeatures (f.brother)} - a \)

\[ \text{parent. status} = = 1 \Rightarrow \forall fb \in f.brother \, fb. \text{status} = = 0 : \text{Include (fb)} \]

- **Friend feature:** when there exists a set of features that requires f, if f is removed, all the features in f.reqed will be removed. The propagation rule is:

✓ Remove (f) \( \Rightarrow \forall fr \in f.reqed : \text{Remove (fr)} \)

Based on the above propagation rules, the feature relationship propagation starting from a specific feature f can be achieved in an iterative manner. At the first propagation step, the inclusion or removal of a specific feature f will lead to its adjacent features, such as parent, child and dependent features, being included or removed. Then at each step, the features that are newly bound (included or removed) at the last step will propagate their inclusions or removals to their adjacent features. Finally, the feature relationship propagation terminates if no features are included or removed in one propagation step.

One example is used to illustrate the process of feature relationship propagation. In the feature model of Figure 3.3, the inclusion of feature F will propagate to a set of its adjacent features: E is removed due to the relationship Sr-1; C is included due to the relationship Rq-3; and A is
included due to the relationship $Sr-I$. Then at step two, the features included or removed at step one, including $E$, $C$ and $A$, will propagate to their adjacent features as well: the removal of $E$ results in the removal of $D$ because of $Rq-2$; and the inclusion of $A$ results in the inclusion of $B$ and $R$ because of $Rq-I$ and $Br-I$ respectively. At step three, the newly included or removed features $D$, $B$ and $R$ will propagate to their adjacent features, but only the inclusion of $R$ results in the inclusion of $K$ by $Br-5$ at this step. As no features will be removed or included because of the inclusion of $K$ at step four, the feature relationship propagation terminates. In conclusion, the feature relationship propagation starting from the inclusion of $F$ in the feature model of Figure 3.3 will lead to the inclusion of $C$, $A$, $B$, $R$, $K$ and the removal of $E$, $D$.

![Diagram of Feature Model](image)

**Figure 3.3 A Feature Model for Illustrating Feature Relationship Propagation**

### 3.3 Detecting Feature Model Errors

This section develops a method of detecting feature model errors by finding all contradictory relationship sets (CRSs) existing in a feature model. For a specific feature $f$, to identify $CRS(f$,
inclusion), the inclusion of f is propagated by feature relationship propagation. Whenever a contradictory configuration conclusion arises, a CRS (f, inclusion) is identified. Similarly, to identify CRS (f, removal), the removal of f is propagated following feature relationship propagation. Whenever a contradictory configuration conclusion arises, a CRS (f, removal) is identified.

The feature relationship propagation is implemented by depth-first search in a feature diagram. The process starts at a specific feature f and the inclusion or removal of f is propagated to the first adjacent feature that can be propagated based on the propagation rules. Then the propagation goes deeper and deeper until a contradictory configuration conclusion is found or until it hits a feature that has no adjacent features that can be propagated based on the propagation rules. Then the search backtracks to the most recent feature it hasn’t finished exploring in a non-recursive implementation, and all freshly propagated features are added to a stack for exploration.

Algorithm 1 is developed to identify all contradictory relationship sets in a feature model. Algorithm 1 includes two functions: the main function (Function 1) and the function of feature relationship propagation (Function 2). Three global variables are defined: crs_list_f which is an array list for storing the identified CRSs starting from f, crs_list which is an array list for storing all CRSs existing in a feature model and path which is a stack for tracking the propagation process. In Function 1, to find CRS (f, inclusion), the inclusion of the root feature r is propagated and then the inclusion of f is propagated. If any contradictory configuration
conclusion arises, a CRS (f, inclusion) can be found (line 1-9). To find CRS (f, removal), the inclusion of root feature r and the parent of f are propagated, and the removal of f is propagated to other features. If any contradictory configuration conclusion arises, a CRS (f, removal) can be found (line 10-19). The parameter of Function 1 is the feature model “fm” from which we aim to find all CRSs. Before identifying CRSs starting from a specific feature f, the feature model must be initialized by the method fm. reset which can set the status of all features in the feature model as “0”. The global variable crs_list_f must be cleared to store the CRSs starting from f by the method crs_list_f. clear (). All CRSs identified will be stored into the array list crs_list as illustrated in line 7 and 17.

Algorithm 1:

Global Variables:

- crs_list_f: the list that stores a set of CRSs starting from f
- crs_list: the list that stores all CRSs in fm
- path: the stack that tracks the feature relationship propagation process starting from f

Function 1: Identifying CRSs

Parameters:

- fm: the given feature model from which we aim to identify CRSs

Function identifyCRS (fm)

    // identify CRS (f, inclusion)
    1. Foreach f in fm. variables
    2.     fm. reset ();
    3.     crs_list_f. clear ();
    4.     propagate (r, 1); // r is the root feature of the feature model
    5.     propagate (f, 1);
Foreach crs_f in crs_list_f
    crs_list.add (new CRS (f, inclusion, crs_f));
End foreach
End foreach

// identify CRS (f, removal)
Foreach f in fm.variables
    fm.reset();
    crs_list_f.clear();
    propagate (r, 1);
    propagate (f.parent, 1);
    propagate (f, -1);
End Foreach
End Foreach

In Function 2, the action imposed on a specific feature f is propagated to other features. This function has three parameters: a feature f where the propagation starts from, an action imposed on feature f where value “1” illustrating inclusion of f and value “-1” illustrating the removal of f, and the relationship that connects f with its precedent feature which propagates value to f. Before the propagation, feature f is included or removed based on the action imposed on f (line 1) and the feature relationship that connects f with its precedent feature is stored into path to track the propagation process (line 2). In the propagation process, if the action on f can be propagated to an adjacent feature, the function “propagate” is called recursively. If an adjacent feature that has already been included needs to be removed or an adjacent feature that has already been removed needs to be included, a contradictory configuration conclusion is found and the propagation returns to the precedent feature (i.e., line 8, 25). Meanwhile the current propagation path is copied and stored to crs_list_f to present the found contradictory
relationship set. If no adjacent features of \( f \) can be removed or included, the propagation returns to the precedent feature and the feature relationship connecting \( f \) and its precedent feature is popped from \( path \) (line 96). It should be noted that after a CRS is found, the propagation will continue as there may be more than one CRS starting from a feature \( f \). After the propagation, all CRSs starting from \( f \) will be stored in \( crs_list_f \).

Algorithm 1:

Function 2: Feature relationship propagation

Parameters:
\( f \): the feature where the propagation starts from in one propagation step.
\( action \): the configuration action imposed on \( f \), “1” illustrating inclusion of \( f \) and “-1” illustrating the removal of \( f \).
\( relationship \): the relationship that connects \( f \) with the feature which propagates the value to \( f \).

Function propagate \( (f, action, relationship) \)

1. \( f. \text{setStatus} \) (action);
2. \( path. \text{push} \) (relationship);

   // propagate inclusion of \( f \) to its adjacent features
3. If \( (action = 1) \)
   // propagate to parent feature
4. If \( (f. \text{parent} != \text{null} && f. \text{parent. status} = 0) \)
5. propagate \( (f. \text{parent}, 1, \text{getRelation} (f, f. \text{parent})) \);
6. End If
7. If \( (f. \text{parent} != \text{null} && f. \text{parent. status} = -1) \)
8. \( crs_list_f. \text{Add} \) \( (path. \text{clone} ()) \);
9. End If

   // propagate to child features
   // \( \text{getNumofFeatures} () \) takes a variation point as input and returns the number of variant
features under the variation point as output.

`getNumofRemoved` takes a variation point as input and returns the number of removed variant features under the variation point as output.

10. If (f is a variation point)
11.    If (getNumofRemoved (f) = getNumofFeatures (f) – f. min_card)
12.      Foreach child in f. children
13.         If (child. status = = 0)
14.          propagate (child, 1, getRelation (f, child));
15.      End If
16.    End Foreach
17. End If
18. Else
19.    Foreach child in f. children
20.     If (child is a mandatory feature or variation point)
21.         If (child. status = = 0)
22.          propagate (child, 1, getRelation (f, child));
23.     End If
24.     If (child. status = = -1)
25.         crs_list_f. Add (path. clone ());
26.     End If
27.    End If
28.    End Foreach
29. End If

//propagate to brother features

`getNumofSelected` takes a variation point as input and returns the number of included variant features under the variation point as output.

30. If (f is variant feature in a variation point && f. parent. status = = 1)
31.    If (getNumofSelected (f. parent) = = f. parent. max_card)
32.      Foreach child in f. parent. children
33.         If (child. status = = 0)
34.          propagate (child, -1, getRelation (f, child));
35.      End If
36.    End Foreach
37. End If
38. Else

//propagate to friend features
Foreach reqs in f. requiresfeatures
    If (reqs. status == 0)
        propagate (reqs, 1, getRelation(f, reqs));
    End If
    If (reqs. status == -1)
        crs_list_f. Add (path. clone ());
    End If
End Foreach

Foreach excs in f. excludesfeatures
    If (excs. status == 0)
        propagate (excs, -1, getRelation(f, excs));
    End If
    If (excs. status == 1)
        crs_list_f. Add (path. clone ());
    End If
End Foreach

End If

// propagate the removal of f to its adjacent features
If (action == -1)
    // propagate to parent feature
    If (f. parent! = null && f is a mandatory feature or variation point)
        If (f. parent. status == 0)
            propagate (f. parent, -1, getRelation (f. f. parent));
        End If
        If (f. parent. status == 1)
            crs_list_f. Add (path. clone ());
        End If
    End If

    // propagate to child feature
    Foreach child in f. children
        If (child. status == 0)
            propagate (child, -1, getRelation(f, child));
        End If
        If (child. status == 1)
            crs_list_f. Add (path. clone ());
        End If
    End Foreach
End If
Using Algorithm 1, all the contradictory relationship sets (CRSs) existing in a feature model can be identified. Based on the relationships between CRSs and feature model errors proposed in
Section 3.1, the feature model errors can be identified based on the found CRSs. For example, in the tourist guide SPL feature model of Figure 2.2, a CRS \(\text{(Mobile, inclusion)}\) can be identified using Algorithm 1, so feature Mobile is a dead feature. To fix this feature model error, the explanations of the feature model error must be given. The next section will develop the method of generating the explanations of a feature model error based on the CRSs that cause the feature model error.

### 3.4 Explaining Feature Model Errors

Once a feature model error is identified, the explanations of the error need to be provided to correct the error. A feature model error may have a set of explanations and each explanation is a set of feature relationships which must be modified to remove the error. For example, a possible explanation of dead feature Mobile in Figure 2.2 would be the feature dependency "Terminal Device requires Encryption", which means that modifying this feature dependency will remove the dead feature error. Another possible explanation of dead feature Mobile would be "Encryption excludes Mobile" and the error can also be corrected by modifying this feature dependency.

The task of explaining a feature model error is a diagnosis problem in the theory of diagnosis (Reiter 1987). In Reiter’s theory, a diagnosis system is modeled as \((SD, COMPS, OBS)\) where

* \(SD\) is a set of predicates defining the behavioral and structural models of the system,
* \(COMPS\) is the set of system components and
* \(OBS\) is a set of observations expressed as predicates.

The abnormal behavior of a component \(c\) is represented as \(Ab(c)\) while the normal behavior of \(c\) is \(N\).
represented as $-Ab(c)$. In Reiter’s theory, $\Delta \subseteq COMPS$ is an explanation of system $(SD, COMPS, OBS)$ if the predicate “$SD \land OBS \land \{Ab(c)|c \in \Delta\} \land \{ -Ab(c) \ | c \in COMPS-\Delta\}$” is consistent and $\Delta$ is minimal. The minimal explanation of a diagnosis system $S$ is defined as an explanation $E$ for $S$ such that no strict subset of $E$ is also an explanation for $S$ (Poole and Mackworth 2010). In the context of feature model validation, the minimal explanations of a feature model error are defined as follows:

**Definition 1:**

*A minimal explanation of a feature model error is an explanation $E$ of the error such that no strict subset of $E$ is also an explanation of the error, where $E$ is a set of feature relationships that must be modified to remove the error.*

In the following process, a method of generating the minimal explanations of a feature model error is developed based on the contradictory relationship sets that cause the error. As a feature model error on a specific feature $f$ is caused by one or more contradictory relationship sets, the intuitive idea of fixing a feature model error on $f$ is to eliminate all the contradictory relationship sets that cause the error from the feature model. Assuming that a feature model error is caused by a set of contradictory relationship sets “$CRS_1, CRS_2…CRS_n$” where “$n=1$” means that the feature model error is caused by one CRS and “$n>1$” means the feature model error is caused by multiple CRSs. As it is known, removing a CRS can be achieved by modifying one or more feature relationships in the CRS. Therefore there can be several solutions of removing a CRS by changing at least one feature relationship in the CRS. Among these solutions, the optimal
solutions of removing a CRS should include one and only one feature relationship in the CRS. Deductively, the optimal solutions of removing “CRS₁, CRS₂…CRSₙ” should include one feature relationship from each CRS and they can be obtained from the Cartesian product of CRS₁, CRS₂…CRSₙ (CRS₁ × CRS₂ × … × CRSₙ = {(r₁, r₂… rₙ): rᵢ ∈ CRSᵢ}). Generating the minimal explanations of a feature model error caused by “CRS₁, CRS₂…CRSₙ”, four steps need to be followed:

- **Step 1**: Obtain all solutions of removing “CRS₁, CRS₂…CRSₙ” by MS = CRS₁ × CRS₂ × … × CRSₙ = {s₁, s₂… sₘ} where m = Card (CRS₁) × Card (CRS₂)×…× Card (CRSₙ).

- **Step 2**: Combine the repeated elements in a solution sᵢ (sᵢ ∈ MS), as an element can only appear once in a solution.

- **Step 3**: Remove the solution sᵢ from MS if another solution sⱼ is the strict subset of sᵢ (sᵢ, sⱼ ∈ MS) based on Definition 1.

- **Step 4**: Use the remaining solutions in MS as the minimal explanations of the feature model error caused by “CRS₁, CRS₂…CRSₙ”.

The dead feature A in feature model of Figure 3.4 is used as an example to illustrate the process of generating the minimal explanations of a feature model error from its related CRSs. First, Algorithm 1 is used to find the CRSs that cause dead feature A: CRS₁ = {Rq-1, Ex-1} and CRS₂...
Then all optimal solutions of removing CRS$_1$ and CRS$_2$ are calculated: $s_1=\{Rq-1, Rq-2, Ex-2\}$, $s_2=\{Rq-1, Rq-1\}$, $s_3=\{Rq-1, Ex-2\}$, $s_4=\{Ex-1, Rq-1\}$, $s_5=\{Ex-1, Rq-2\}$, $s_6=\{Ex-1, Ex-2\}$ by $CRS_1 \times CRS_2$. As $s_1$ has repeated elements, it is combined into $\{Rq-1\}$. The solutions $s_2$, $s_3$ and $s_4$ are removed from the solution list as $s_1=\{Rq-1\}$ is their strict subset. Finally, the remaining solutions $s_1=\{Rq-1\}$, $s_5=\{Ex-1, Rq-2\}$ and $s_6=\{Ex-1, Ex-2\}$ can be used as the minimal explanations of dead feature $A$.

**Figure 3.4 An Example for Illustrating Minimal Explanations**

In Reiter’s theory, the error explanation for dead feature $A$ in Figure 3.4 can be represented as a diagnosis system $(SD’, COMPS’, OBS’)$ where the predicate “$SD’ \land OBS’ \land \{Ab(c’) \mid c’ \in \Delta\} \land \{ – Ab(c’) \mid c’ \in COMPS’ \backslash \Delta\}$”. Here, $SD’$ includes the rules of constraining feature selections in the feature model and these rules arise from all feature relationships; $COMPS’$ includes all feature relationships in the feature model and $OBS’$ indicates the inclusions or removals of some specific features. $Ab (c’)$ means that feature relationship $c’$ should be absent in the feature model.
and $\neg Ab \ (c')$ means that the feature relationship $c'$ should be present in the feature model. To find the explanations of dead feature $A$, the predicate in $OBS'$ would be the inclusion of feature $A$. As the found minimal explanations of dead feature $A$: $\Delta = \{Rq-1\}, \Delta = \{Ex-1, Rq-2\}$ and $\Delta = \{Ex-1, Ex-2\}$ can make the predicate $"SD' \land OBS' \land [Ab(c')|c' \in \Delta] \land \neg Ab(c') |c' \in COMPS'-\Delta"$ consistent, it can be said that the minimal explanations identified by this proposed validation approach are the right explanations of the feature model error in Reiter’s theory. Table 3.2 uses some examples to illustrate the process of generating the minimum explanations for a feature model error based on the contradictory relationship sets that cause the feature model error.

<table>
<thead>
<tr>
<th>Table 3.2 Examples of Generating Minimal Explanations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature Model Error and CSR</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>dead feature $B$</td>
</tr>
<tr>
<td>CRS (B, inclusion)</td>
</tr>
<tr>
<td>CRS (B, inclusion)</td>
</tr>
<tr>
<td>dead feature $B$</td>
</tr>
<tr>
<td>CRS (B, inclusion)</td>
</tr>
<tr>
<td>CRS (B, inclusion)</td>
</tr>
</tbody>
</table>

3.5 Concluding Remarks

This chapter proposes a feature model validation approach based on the contradictory feature relationships behind feature model errors. As the algorithm of finding the contradictory feature
relationships from a feature model in this approach is the same as finding the elementary circuits from a directed graph (Tarjan 1973), the time complexity of this proposed approach can have an upper bound of $O(v + e)(c + 1)$ when applied to a feature model with $v$ features, $e$ feature relationships and $c$ contradictory relationship sets (Johnson 1975). As the existing CSP-based validation approaches have the time complexity $O(2^n)$ in the general cases, this proposed validation approach is more efficient than the existing approaches if the number of CRSs in a feature model is not as large as $2^n$. An empirical evaluation of our validation approach will be presented in Chapter 6.

Currently, the validation approach proposed in this chapter can be used to validate the cardinality-based feature models which include three kinds of hierarchical relationships: mandatory, optional and feature group, and two kinds of cross-tree constraints: requires and excludes. To debug a feature model with other types of feature constraints, this validation approach needs to be extended. As introduced earlier, the impact relationships (Zhang et al. 2005) cannot constrain the selection among features, but can imply the positive or negative impact of the configuration decision of a specific feature on other features. In order to deal with the impact relationships in this proposed validation approach, the propagation rules for the impact relationships need to be defined. Furthermore, the rules of triggering the selection or removal of a specific feature need to be set based on the accumulated impact from other features. In this way, this validation approach can be extended to debug a feature model with impact relationships.
The feature model validation approach is the prerequisite for modeling quality attributes in a feature model in the quality framework proposed in this thesis. Without knowing the validity of a feature model, the quality attributes may be modelled for an invalid feature model. If the invalid feature model is corrected for product configuration, the previously captured quality knowledge cannot be used to assess the product quality for the new feature model. In this case, extra efforts need to be paid to model the quality attributes in the new feature model. Therefore, the validation approach proposed in this chapter can be considered as the prerequisite of the quality modeling approach proposed in next chapter.
4. An Approach for Quality Attributes Modeling in a Feature Model

This chapter proposes a systematic approach of modeling quality attributes in a feature model and enhancing feature-based product configuration based on the captured quality knowledge. To avoid generating real products, domain experts’ judgments are used to measure the interdependencies between functional features and quality attributes. To reduce the human effort involved in the judgments, the analytic hierarchical process is used to estimate the relative impacts of individual features on a quality attribute. Once the interdependencies have been recognized, this captured knowledge can be used to support quality-aware product configuration.

4.1 Overview

The whole process of the quality attributes modeling approach is depicted in Figure 4.1, including the major phases with related artefacts. In Figure 4.1, the arrows represent the production and consumption relations between phases and artifacts and also the order of phases and activities. There are three major phases in this approach: 1) identify and represent quality attributes in feature models, 2) measure the interdependencies between features and quality attributes, and 3) quality-aware product configuration. Phases 1) and 2) aim to model quality attributes in feature models in domain engineering while 3) aims to configure a product in a quality-aware manner in application engineering.

The starting phase 1) identifies the quality attributes that are critical for a software product line by adapting non-functional requirement (NFR) framework (Chung et al. 2000) and extends the
current feature models with a sub-feature tree to represent the identified quality attributes. Based on the extended feature model, phase 2) measures the interdependencies between features and the identified quality attributes based on domain experts’ judgments. Two important issues are addressed in phase 2): first, the relative impacts of individual features on a quality attribute are estimated by using an analytic hierarchical process (Hallowell and David 2007) which is a pairwise comparison method; second, the relationships among the features with respect to affecting a quality attribute are identified to quantify the aggregated impact of a configured product. The output of phase 2) is a quality attribute knowledge base (QAKB) which is a reusable artifact to store the captured quality knowledge. Based on the QAKB, phase 3) conducts quality-aware product configuration, which is the process of guiding the application engineers to derive a desired product that satisfies the functional needs and the quality requirements. The following sections introduce each phase of this proposed approach in detail.
4.2 Identify and Represent Quality Attributes in Feature Models

The first phase of the quality modeling approach is to identify the quality attributes that are critical for a software product line and represent the identified quality attributes in feature models. An adapted NFR framework (Chung et al. 2000) is used to identify quality attributes for a software product line and extend the current feature models with quality attribute (QA)
features to represent the identified quality attributes.

The NFR framework is a process-oriented approach to deal with the non-functional requirements (NFRs) (Mylopoulos et al. 1992). It provides a systematic way of modeling and analyzing NFRs in software development process (Chung et al. 2000). This framework supports identifying and representing NFRs explicitly, identifying design alternatives which could meet the stated NFRs, exploring the contributions from design alternatives to stated NFRs and selecting among design alternatives in terms of best satisfying the certain quality attribute. In the NFR framework, a goal model is used to represent NFRs, design alternatives and the contribution relationships between them (Giorgini et al. 2002). In a goal model, the non-functional requirements are denoted by NFR softgoals and design techniques are represented as operationalising softgoals. Figure 4.2 shows a softgoal model from banking systems. A set of correlation links (make, help, unknown, hurt and break) are defined to describe the contributions of operationalising softgoals on NFR softgoals.
In a goal model, the abstract NFRs such as security, usability and performance are represented as the top-level softgoals. These softgoals are vague and difficult for stakeholders to understand and they are always refined into more detailed sub-softgoals which have more semantics by AND/OR refinements. The softgoal refinement will eventually reach some operationalising softgoals which are design techniques for satisfying NFR softgoals. In Figure 4.2, two abstract NFR softgoals security and performance are refined into a set of sub-softgoals and finally reach three operationalising softgoals password, key and biometric. The selection from these three design alternatives will result in different levels of security and performance through propagation algorithm (Baixauli et al. 2004; Giorgini et al. 2002; Lamsweerde 2009).
The NFR framework provides two kinds of catalogues to support softgoal decomposition: NFR sort catalogue and NFR operationalization method catalogue based on the development knowledge taken from the literature and industrial experiences. A NFR sort catalogue summarizes the potentially set of concepts of the NFR and organized into a hierarchy. Figure 4.3 shows a NFR sort catalogue for security and performance. The NFR sort catalogue can assist the developers in refining abstract NFR softgoals into detailed NFR softgoals. A NFR operationalization method catalogue concludes the design techniques for satisfying the specific NFR. Figure 4.4 shows an operationalization method catalogue for security. The NFR operationalization method catalogue can help refine the NFR softgoals into operationalising softgoals.

Figure 4.3 A Portion of Sort Catalogue of Security and Performance
By adapting the NFR framework and its goal oriented models, three sub-steps need to be followed to identify and represent quality attributes in feature models:

1. identify the critical quality attributes for a software product line
2. refine the quality attributes into sub-quality attributes
3. represent quality attributes as features in a feature model

Firstly, the most critical quality attributes for a software product line (SPL) are identified by adapting NFR framework (Chung et al. 2000). It is a difficult and time-consuming task for domain experts to identify the critical quality attributes for software systems in an SPL because of the elusive nature of quality attributes. The NFR template in NFR framework provides a detailed classification and description for each specific quality attribute. In the NFR template, quality attributes are classified into system qualities and system constraints. System qualities are
properties or characteristics of the software systems that the stakeholders care about and hence will affect their degree of satisfaction with the systems. Examples include usability, supportability, availability, reliability, performance, security, scalability, modifiability, resolvability and reusability. System constraints are the characteristics of development environment or organization that constrain the development in some way. Examples include the target operating system or hardware platform in the case of user environment, or the skill of available developers in the case of development organization. In this proposed approach, the NFR template is used as a checklist to identify the most critical quality attributes for a software product line.

Secondly, the quality attributes identified from the NFR template need to be refined into detailed quality attributes using NFR decomposition methods proposed in the NFR framework. The quality attributes identified from the NFR template are always abstract. The degree of specificity of the identified quality attributes, such as security and performance, would not permit the non-functional requirements analysis, and the abstract quality attributes need to be refined into more detailed quality attributes which have more semantics. An NFR framework provides a sort catalogue for each abstract quality attribute based on the development knowledge taken from the literature and industrial experiences. An NFR sort catalogue summarizes the potential set of concepts (sorts) of the quality attribute in a hierarchy and can serve as a rich set of alternatives to choose from and serve as check-points to guard against omitting any important concerns in quality attribute decomposition. If a sub-quality attribute in the NFR sort catalogue is critical for one or more product line members, it will be included into
the decomposition; otherwise it will be excluded from the decomposition.

Finally, the abstract quality attributes and the refined sub-quality attributes need to be represented in a feature model. Kang et al. (1990) defined a feature as “the prominent or distinctive user-visible aspect, quality or characteristic of a software system or systems”. Based on this definition, each quality attribute can be modelled as a feature. The current feature models are extended with a sub-feature tree to represent the quality attributes. This sub-feature tree is named as quality attribute (QA) feature tree and its included features are named as QA features. The root of the QA feature tree is the overall quality attribute which represents the overall goodness of the system. The second level of the QA feature tree is formed by the abstract quality attributes identified from an NFR template. For example, performance, security, usability and availability are the children of the root. Each of these quality attributes has specific quality attribute refinements. The leaves of the QA feature tree are the detailed sub-quality attributes that are concrete enough for prioritization and analysis. The QA feature tree has mandatory and optional as hierarchical relationships, as these two relationships are sufficient for describing the relationships between an abstract quality attribute and its refined sub-quality attributes. The relationships between functional features and QA features, as well as the relationships among QA features, cannot be simply represented by feature dependencies (i.e. requires and excludes) which are used to constrain the selection among functional features.

Following the above three steps, three abstract quality attributes: performance, low cost and security, are identified from the tourist guide software product line of Figure 2.2 and refined into
sub-quality attributes. The original feature model is extended with a sub-quality attribute feature tree as illustrated in Figure 4.5.

![Figure 4.5 The Extended Feature Model of Tourist Guide Software Product Line](image)

### 4.3 Measure Interdependencies between Features and Quality Attributes

Once the quality attributes of a software product line are identified and represented as quality attribute (QA) features in an extended feature model, the interdependencies between features and QA features can be measured, such as evaluating the relative impact of different features on a quality attribute using analytic hierarchical process (AHP) (Hallowell and David 2007) and estimating the collective impact of these contributions on a quality attribute based on the defined quantitative assessment. A set of sub-steps needs to be followed to measure the interdependencies in a feature model.
- **Step 1: Identify the contributors of a quality attribute (QA):** A quality attribute (QA) is related to a set of features whose inclusions or exclusions will have either positive or negative impact on QA. The set of features affecting a quality attribute are named as contributors of the quality attribute. The first step of measuring interdependencies is to identify the contributors of a quality attribute QA.

- **Step 2: Prioritize the contributors of QA:** The identified contributors have different impacts on QA. The second step is to prioritize the identified contributors based on their relative impact on QA by using AHP.

- **Step 3: Identify the relationships among the contributors of QA:** The contributors of QA often affect QA interdependently. The third step is to identify the relationships among the identified contributors with respect to affecting a quality attribute.

- **Step 4: Calculate the overall impact on QA:** The QA level of a configured product is determined by the overall impact of the set of contributors of QA included in the configured product. The fourth step is to calculate the overall impact of a set of contributors of QA based on the impact of individual contributors and the relationships among these contributors.
• **Step 5: Normalize the overall impact on QA:** The calculated overall impact of a configured product on QA cannot represent its relative QA level in the application domain comparing with other product line members. The fifth step is to normalize the overall impact of a configured product to represent its relative QA level.

• **Step 6: Check the consistency of domain experts’ judgment:** Domain experts may make wrong judgments which will lead to incorrect quality attribute prediction for a configured product. Therefore the measured interdependencies need to be validated by checking the consistency of domain experts’ judgments.

• **Step 7: Represent the interdependencies between QA and its contributors:** The interrelationships between QA and its contributors are complex. The seventh step is to develop a representation schema to represent the measured interdependencies.

• **Step 8: Represent the relationships among related quality attributes:** Some quality attributes are related with each other in the context of software product lines. This kind of relationships among related quality attributes plays an important role in product configuration process. The final step aims to identify the relationships among related quality attributes based on the measured interdependencies.
4.3.1 Identifying Contributors

The identification of the contributors of a quality attribute is mainly based on domain experts’ knowledge and experience. In a feature model, some functional features are the implementation techniques or domain technologies for satisfying a quality attribute and these features can be considered as the contributors of the quality attribute directly. For example, the functional features *password*, *biometrics* and *card key* are the implementation techniques for quality attribute data access security (DAS) and these features can be considered as the contributors of DAS. As mentioned earlier, the NFR framework provides an operationalization method catalogue which concludes the design techniques for satisfying each abstract quality attribute (Chung et al. 2000). This catalogue can be used as a checklist to identify the contributors of a quality attribute. However, not all contributors of a quality attribute can be identified from this catalogue. Some operating-environment features, such as hardware or software platforms, also contribute to a quality attribute, even though they are not included in the operationalization method catalogue for the quality attribute. In this case, domain experts’ knowledge and experience play an important role in the identification process. For example, domain experts know that types of network, terminal devices selected, and encryption of data will have impact on the quality attribute data transfer speed (DTS) in tourist guide SPL. These functional features cannot be identified as contributors from the operationalization method catalogue for performance. The domain experts can identify these contributors based on their experience and knowledge. Following the above method, the contributors of DTS in the tourist guide software product line in Figure 4.5 have been identified as *Encryption, PDA, Mobile, Modem19200, Modem9600, LAN* and *WAN*. 
The identified contributors of a quality attribute may have either a positive or negative impact on the quality attribute. For example, the feature *Encryption* has a positive impact on data transfer security (DTSS) while it has a negative impact on DTS when it is included into the product. To be convenient for AHP pair-wise comparisons, the impact type needs to be unified.

If a feature has a negative impact on a quality attribute when it is included into the product (included status), it has a positive impact on the quality attribute when it is excluded from the product (removed status). This means that a negative impact on a quality attribute made by a variable feature in included status can be transformed into a positive impact on the quality attribute in removed status. $RF (QA, +)$ is used to represent the set of features which have a positive impact on $QA$ when they are included into the product and $RF (QA, -)$ is used to represent the set of features which have a positive impact on $QA$ when they are excluded from the product. Further $RF (QA) = RF (QA, +) \cup RF (QA, -)$ is used to represent all the contributors of $QA$. Among the set of contributors of DTS, it is found that *Encryption* has a negative impact on DTS in included status. The negative impact of *Encryption* in included status can be transformed to a positive impact of *Encryption* in removed status. Following the above definitions, the contributors of DTS can be represented as: $RF (DTS, +) = \{LAN, WAN, PDA, Mobile, Modem 9600, Modem 19200\}$, $RF (DTS, -) = \{Encryption\}$ and $RF (DTS) = \{LAN, WAN, PDA, Mobile, Modem 9600, Modem 19200, Encryption\}$. 
4.3.2 Prioritizing Contributors

Once the contributors of a quality attribute have been identified, these contributors need to be prioritized based on their relative importance for satisfying the quality attribute. The analytic hierarchical process (AHP) (Saaty 2001, 2008), a pair-wise comparison method, is used to measure the relative impact of the contributors of a quality attribute $QA$. The AHP is chosen as the comparison technique for two reasons. First, AHP is one of the most promising methods for prioritization according to evaluation criteria (Hallowell and David 2007). AHP is often used for the purpose of making relative assessments, such as relatively comparing all the involved elements according to their importance based on stakeholders’ preferences. This characteristic of AHP meets the need of comparing the relative impact of individual features on a quality attribute in the proposed approach. Second, the key issue of AHP is the paired comparison which occurs between each two elements among all involved elements. Using AHP, it is much easier for domain experts to make judgments on the relative importance of a pair of features.

In the AHP method, domain experts use a value in [-9.0, +9.0] scale to represent the relative importance of the two elements in a pair-wise comparison based on the intensity of importance as shown in Table 4.1. A comparison matrix, which uses all the involved elements as both the column members and the row members, is made based on the pair-wise comparisons. If the element in the row is more important than the member in the column, the actual intensity of importance value is assigned, whereas if the element in the row is less important than the element in the column, the negative of the corresponding intensity of importance value is assigned. Based on the comparison result in the matrix, an AHP tool can be used to calculate a
priority vector which consists of the relative importance value of each involved element. A consistency ratio is also calculated to represent the consistency of all pair-wise comparisons. A value below 0.1 is proof of good consistency.

<table>
<thead>
<tr>
<th>Intensity of Importance</th>
<th>Definition</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Equal importance</td>
<td>Two elements contribute equally to the objective.</td>
</tr>
<tr>
<td>3</td>
<td>Moderate importance</td>
<td>Experience and judgment slightly favor one element over another.</td>
</tr>
<tr>
<td>5</td>
<td>Strong importance</td>
<td>Experience and judgment strongly favor one element over another.</td>
</tr>
<tr>
<td>7</td>
<td>Very strong importance</td>
<td>One element is favored very strongly over another, its dominance is demonstrated in practice.</td>
</tr>
<tr>
<td>9</td>
<td>Extreme importance</td>
<td>The evidence favoring one element over another is of the highest possible order of affirmation.</td>
</tr>
</tbody>
</table>

In this quality modeling approach, domain experts use the AHP method to compare each pair of features in $RF (QA)$ and assign a value to each comparison. This value represents domain experts’ belief about how much a feature is more important or less important than another feature in a pair-wise comparison. This value comes from either objective data or domain experts’ experience accumulated from the development of similar projects. Two domain experts are used to make judgments to ensure that they make consistent and unbiased judgments. The detailed method of evaluating domain experts’ judgments will be described in Section 4.3.6. Based on the pair-wised comparisons, a comparison matrix is generated, and the relative impact of each feature in $RF (QA)$ is calculated from the comparison matrix using AHP tools. The
Relative Importance Value (RIV) is defined to be the relative impact of each individual contributor on QA, and RIV (QA, f) is used to represent the RIV of feature f on QA. If \( f \in RF (QA, +) \), RIV (QA, f) represents the RIV of feature f on QA when f is selected while if \( f \in RF (QA, -) \), RIV (QA, f) represents the RIV of feature f on QA when f is excluded.

The following example is used to illustrate the process of using the AHP to calculate the relative impact of individual contributors on QA. The set of contributors of data transfer speed (DTS) in the tourist guide software product line have been identified earlier. To calculate the relative impact of each contributor on DTS, domain experts compare each pair of features from RF (DTS) and assign a value to each comparison according to the intensity of importance in Table 4.1. Then, the comparison results are written into a matrix as illustrated in Figure 4.6. For example, Modem 19200 in the row is moderately more important than the Mobile in the column, so a value of “+5” is assigned to the corresponding cell of the matrix. Mobile in the row is absolutely less important than the LAN in the column, so a value of “-9” is assigned. Finally, a priority vector can be identified from the comparison matrix and this priority vector consists of the RIV of each contributor on the quality attribute DTS as illustrated in Figure 4.7. The method used to calculate the RIV is to calculate the normalized Eigen vector of the comparison matrix (Hallowell and David 2007). For example, the RIV of individual contributors in RF (DTS) have been calculated and it can be known that RIV (DTS, Encryption) = 15.45 and RIV (DTS, LAN) = 33.37.
4.3.3 Identifying Feature Relationships

Once the impact of individual contributors of QA has been calculated, the overall impact on QA made by a set of contributors from RF (QA) can be calculated. The Overall Importance Value (OIV) is defined to be the overall impact of a combination of contributors and \( OIV (QA, fg) \) is used to represent the OIV of a set of contributors \( (fg) \) which is a subset of RF (QA) \( (fg \subset RF) \).
(QA). Intuitively, the simplest way to calculate $OIV(QA, fg)$ is to add the relative importance values of all the contributors in $fg$. However, in many cases, some contributors of $QA$ will affect $QA$ interdependently, i.e. they are related with each other in terms of affecting $QA$. The overall impact of two related contributors may not be the sum of their relative impact. For example, $password$ and $key$ are two contributors of quality attribute data access security (DAS). When they are included into the target product for satisfying DAS, it is not rational to add the $RIV$ of $password$ and the $RIV$ of $key$ together as the $OIV$ for the inclusion of both $password$ and $key$, because the overall impact on DAS is likely the maximum value of the two contributors. To recognize the inter-relationships among some contributors in $RF(QA)$ in terms of affecting $QA$, the following four types of feature groups are defined:

- **SumGp**: If the $OIV$ on $QA$ made by the selected contributors from a feature group can be considered as the sum of the $RIV$s of the individual selected contributors, the feature group is called SumGp.

- **AvgGp**: If the $OIV$ on $QA$ made by the selected contributors from a feature group can be considered as an average of the $RIV$s of individual selected contributors, the feature group is called AvgGp.

- **MaxGp**: If the $OIV$ on $QA$ made by the selected contributors from a feature group can be represented by the maximum $RIV$ among all the selected contributors, the feature group is called MaxGp.
• **MinGp**: If the *OIV* on *QA* made by the selected contributors from a feature group can be represented by the minimum *RIV* among all the selected contributors, the feature group is called **MinGp**.

The above definitions can cover the most common feature group types. However, there may be other types of feature groups which exhibit more complex feature relationships in terms of affecting a quality attribute. In this case, domain experts can define domain-specific feature groups. Following the above defined feature groups, domain experts identify feature groups from *RF* (*QA*) based on their domain experience and knowledge accumulated from similar projects. In the case of tourist guide software product line, all feature groups for satisfying DTS are identified from *RF* (*DTS*) as follows:

\[
\begin{align*}
fg_1 (DTS) &= \{Encryption\}, \text{SumGp} \\
fg_2 (DTS) &= \{Modem 19200, Modem 9600\}, \text{AvgGp} \\
fg_3 (DTS) &= \{PDA, Mobile\}, \text{AvgGp} \\
fg_4 (DTS) &= \{LAN, WAN\}, \text{MinGp}
\end{align*}
\]

### 4.3.4 Calculating the Overall Impact

A feature configuration, either a full configuration or a partial configuration, may only include a subset of *RF* (*QA*) for satisfying *QA*. The *valid selection* of a feature configuration with respect to affecting *QA* is defined to be the set of contributors of *QA* included in the feature
configuration and VS (QA, FC) is used to represent the valid selection with respect to quality attribute QA in a feature configuration FC. Then the overall impact of a feature configuration FC on QA can be considered as the overall importance value of its included valid selection OIV (QA, VS (QA, FC)) which can be calculated based on different feature groups. The calculation of OIV (QA, VS (QA, FC)) is illustrated as follows. Assume that RF (QA) can be divided into a set of disjoint feature groups: \(fg_1, fg_2, \ldots, fg_n\) and \(fg_1 \cap fg_2 \cap \ldots \cap fg_i \cap \ldots fg_n = \emptyset\). The valid selection of a feature configuration VS (QA, FC) can also be divided into a set of disjoint groups: \(vfg_1, vfg_2, \ldots, vfg_n\) and \(vfg_1 \subseteq fg_1, vfg_2 \subseteq fg_2, \ldots, vfg_n \subseteq fg_n\). Then OIV (QA, VS (QA, FC)) can be calculated as \(\text{Sum} (\text{OIV} (QA, vfg_i))\). The OIV (QA, vfg_i) can be calculated based on the following formulas which show how to calculate the overall importance value of a set of contributors in each type of feature group.

- \(\text{OIV} (QA, vfg_i) = \text{Sum} (RIV (QA, f_j) \mid f_j \in vfg_i, vfg_i \subseteq fg_i)\), if \(fg_i\) is a \(\text{SumGp}\).
- \(\text{OIV} (QA, vfg_i) = \text{Average} (RIV (QA, f_j) \mid f_j \in vfg_i, vfg_i \subseteq fg_i)\), if \(fg_i\) is an \(\text{AvgGp}\).
- \(\text{OIV} (QA, vfg_i) = \text{Maximum} (RIV (QA, f_j) \mid f_j \in vfg_i, vfg_i \subseteq fg_i)\), if \(fg_i\) is a \(\text{MaxGp}\).
- \(\text{OIV} (QA, vfg_i) = \text{Minimum} (RIV (QA, f_j) \mid f_j \in vfg_i, vfg_i \subseteq fg_i)\), if \(fg_i\) is a \(\text{MinGp}\).

For example, \{Tourist Guide, Security Mechanism, Operating Environment, Network Connection, WAN, Terminal Device, PDA, Mobile, Position Detection, Satellite, Web-Based\} is a product configured from the feature model of Figure 4.5. As the contributors of data transfer speed (DTS) are \(RF (DTS, +) = \{LAN, WAN, PDA, Mobile, Modem 9600, Modem19200\}\) and \(RF (DTS, -) = \{Encryption\}\), the valid selection VS with respect to DTS can be identified as
{WAN, Mobile, PDA, Encryption}. Then VS is divided into four groups: $vfg_1$, $vfg_2$, $vfg_3$, $vfg_4$ where $vfg_i = f_i \cap VS$. Based on the above formulas, the $OIV$ of each group as well as the $OIV$ of VS can be calculated. The calculation process and results are illustrated in Table 4.2.

<table>
<thead>
<tr>
<th>Feature Groups</th>
<th>Members of Features</th>
<th>Group Type</th>
<th>$OIV\ (DTS)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$vfg_1$</td>
<td>{Encryption}</td>
<td>SumGp</td>
<td>15.45</td>
</tr>
<tr>
<td>$vfg_2$</td>
<td>$\emptyset$</td>
<td>AvgGp</td>
<td>0</td>
</tr>
<tr>
<td>$vfg_3$</td>
<td>{PDA, Mobile}</td>
<td>AvgGp</td>
<td>2.93</td>
</tr>
<tr>
<td>$vfg_4$</td>
<td>{WAN}</td>
<td>MinGp</td>
<td>28.56</td>
</tr>
<tr>
<td>VS ($DTS$)</td>
<td></td>
<td></td>
<td>46.94</td>
</tr>
</tbody>
</table>

### 4.3.5 Normalizing the Overall Impact

The definitions of quality level of software systems such as high or low depend on a specific domain’s industrial norm and it varies from domain to domain. Therefore, when assessing the quality level for a member product in a software product line (SPL), the customers focus on evaluating its relative level in the application domain compared with other member products in the SPL. The early calculated overall importance value ($OIV$) cannot represent this relative quality level of the configured product. It must be compared with the $OIV$s of all other SPL members to represent its relative quality level. To find the $OIV$s of all member products, it is not necessary to derive all member products and calculate the $OIV$ for the valid selection of each product, as many configured products share the same valid selection. Therefore, it is necessary to find the potential valid selections of all SPL members for a quality attribute. Herein, “VS$^i$
(QA), VS₁(QA)... VSn(QA)” are used to represent the potential valid selections of SPL members for quality attribute QA. Each configured product will include one of these valid selections and each of these valid selections will be included by several configured products. To derive these valid selections, a feature diagram which includes all the features in RF (QA) and the selection constraints among the included features is drawn based on the original feature diagram. From such a feature diagram, all the possible valid selections of SPL members with respect to quality attribute QA can be derived.

![Feature Diagram for the Contributors of Data Transfer Speed](image)

**Figure 4.8 A Feature Diagram for the Contributors of Data Transfer Speed**

The DTS in the tourist guide SPL is used as an example to show how to identify the possible valid selections of all members in the tourist guide SPL. Figure 4.8 shows two kinds of selection constraints among the features in RF (DTS). The multiplicity constraints and cross-tree constraints are extracted from the original feature diagram of Figure 2.2 and the constraint matrix of Table 2.2. Based on the constraints shown in Figure 4.8, all the valid selections of
SPL members with respect to DTS can be derived as shown in Table 4.3. Following the process of calculating the $OIV(DTS, VS)$, the $OIV$ of all the valid selections of DTS can be calculated and the results are also shown in Table 4.3.

<table>
<thead>
<tr>
<th>Valid Selections from RF(DTS)</th>
<th>$OIV$ for $DTS$</th>
<th>$NOIV$ for $DTS$</th>
</tr>
</thead>
<tbody>
<tr>
<td>VS$_1$ {LAN, Modem9600}</td>
<td>55.70</td>
<td>0.89</td>
</tr>
<tr>
<td>VS$_2$ {LAN, Modem19200}</td>
<td>58.69</td>
<td>1.0</td>
</tr>
<tr>
<td>VS$_3$ {LAN, Modem 9600, Encryption}</td>
<td>40.25</td>
<td>0.32</td>
</tr>
<tr>
<td>VS$_4$ {LAN, Modem19200, Encryption}</td>
<td>43.24</td>
<td>0.43</td>
</tr>
<tr>
<td>VS$_5$ {WAN, Mobile}</td>
<td>46.77</td>
<td>0.56</td>
</tr>
<tr>
<td>VS$_6$ {WAN, PDA}</td>
<td>47.11</td>
<td>0.57</td>
</tr>
<tr>
<td>VS$_7$ {WAN, PDA, Encryption}</td>
<td>31.66</td>
<td>0.0</td>
</tr>
<tr>
<td>VS$_8$ {WAN, Mobile, PDA}</td>
<td>46.94</td>
<td>0.57</td>
</tr>
</tbody>
</table>

Finally, the overall importance value of a valid selection is normalized into a value in $[0.0, 1.0]$ scale to represent its relative $QA$ level. The following formula can be used to calculate the $Normalized Overall Important Value$ ($NOIV$) for each valid selection with respect to $QA$.

$$NOIV(QA, VS) = \frac{OIV(VS) - MIN(OIV(VS_i))}{MAX(OIV(VS_i)) - MIN(OIV(VS_i))}$$

In this formula, $MIN (OIV (VS_i))$ is the minimum $OIV$ among all the valid selections of $QA$ and $MAX (OIV (VS_i))$ is the maximum $OIV$ among all the valid selections of $QA$. Using this formula, the $NOIV$ for each valid selection can be calculated as shown in Table 4.3. The minimum value of $NOIV (QA, VS)$ is 0.0 which means that the valid selection VS achieves the lowest $QA$ level in
terms of all SPL members while the maximum value of $NOIV (QA, VS)$ is 1.0 which means that the valid selection $VS$ achieves the highest $QA$ level in terms of all SPL members. A $NOIV$ between 0.0 and 1.0 represents its relative $QA$ level compared with the highest one and the lowest one. The $NOIV$s of valid selections for DTS in the tourist guide SPL are shown in Table 4.3. The $NOIV$ of $VS_8$ is 0.57, which means that any configured product that includes $VS_8$ has a relative medium level of data transfer speed.

### 4.3.6 Evaluating Domain Experts’ Judgments

Domain experts’ judgments are needed in this AHP-based approach. If there are errors in domain experts’ judgments, the relative importance values ($RIV$s) of individual contributors of a quality attribute will be incorrect. Based on the incorrect $RIV$s, the predicted quality level for a configured product will be wrong. Therefore, one critical issue of this proposed approach is to ensure the correctness of domain experts’ judgments.

In the AHP-based approach, domain experts need to make judgments on $n$ $(n-1)/2$ pair-wise comparisons for $n$ contributors of quality attribute $QA$, “$X_1, X_2,…X_n$” based on their contribution to $QA$. The results of pair-wise comparisons are written into a comparison matrix as shown in Table 4.4 where $C_{ij}$ represents the importance intensity value of comparing $X_i$ with $X_j$. A priority vector ($PV$) which consists of the relative importance values of all the contributors of $QA$ involved in the comparison can be calculated from the comparison matrix. Then the relative importance values in $PV$ are used to measure the interdependencies between $QA$ and its contributors. Based on the interdependencies, the $QA$ level for any product configuration can be
Table 4.4 Comparison Matrix for the Contributors of QA

<table>
<thead>
<tr>
<th></th>
<th>$X_1$</th>
<th>$X_2$</th>
<th>$X_{n-1}$</th>
<th>$X_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_1$</td>
<td>1</td>
<td>$C_{12}$</td>
<td>$C_{1(n-1)}$</td>
<td>$C_{1n}$</td>
</tr>
<tr>
<td>$X_2$</td>
<td></td>
<td>1</td>
<td>$C_{2(n-1)}$</td>
<td>$C_{2n}$</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$X_{n-1}$</td>
<td></td>
<td></td>
<td>1</td>
<td>$C_{(n-1)n}$</td>
</tr>
<tr>
<td>$X_n$</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

When making pair-wise comparisons in the AHP method, domain experts may make two kinds of errors. Firstly, a domain expert may make inconsistent pair-wise comparisons for the contributors of QA. For example, in the comparison matrix of Table 4.4, if $(C_{ij} > 0) \land (C_{jk} > 0) \land (C_{ik} < 0)$, there will be conflicts among these three comparisons, because it can be deducted that $C_i$ is more important than $C_k$ as $C_{ij} > 0$ illustrates that $C_i$ is more important than $C_j$ and $C_{jk} > 0$ illustrates that $C_j$ is more important than $C_k$. However, this deduction conflicts with $C_{ik} < 0$ which means $C_i$ is less important than $C_k$. In this case, it can be said that the domain expert makes inconsistent pair-wise comparisons.

Inconsistencies in one domain expert’s judgments can be identified by checking the consistency ratio (CR) of the comparison matrix. AHP allows small inconsistency in judgments because
human judgments are not always consistent. A CR below 0.1 is acceptable (Hallowell and David 2007). The “0.1” is also adapted as a borderline to check whether all the pair-wise comparisons made by one domain expert are consistent. The calculation of CR is supported by most AHP tools. In this approach, if CR of a comparison matrix is above 0.1, the domain expert needs to identify the inconsistent pair-wise comparisons and modify the comparison matrix until the CR is below 0.1.

The second kind of errors that a domain expert may make is the biased judgments. For example, a feature X has significant contribution to a quality attribute QA in one domain expert’s opinion. However, X is not that important to QA in reality. The RIV (QA, X) calculated based on the domain expert’s judgment must be higher than its real value. To avoid the biased judgments made by one domain expert, two domain experts are employed in this approach and the consistency between two domain experts’ judgments are measured. Assume that two domain experts generate two comparison matrixes: matrix (1) and matrix (2) for the feature set “X₁, X₂,...Xₙ” and calculate two priority vectors PV₁ and PV₂ respectively. The NOIV₁ (QA, VSᵢ) illustrates the NOIV of valid selection VSᵢ based on PV₁ while the NOIV₂ (QA, VSᵢ) illustrates the NOIV of valid selection VSᵢ based on PV₂. Then the following formula is used to measure the consistency between two domain experts’ judgments.

\[ AD = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (NOIV₁(QA, VSᵢ) - NOIV₂(QA, VSᵢ))^2} \]
The above formula represents the *average difference* (AD) between the predicted QA levels based on $PV_1$ and predicted QA levels based on $PV_2$ for all valid selections $VS_i$ from $RF (QA)$. The minimum value of AD is 0.0 when $PV_1$ and $PV_2$ are completely the same while the maximum value of AD is 1.0 when $PV_1$ and $PV_2$ are completely contrary. A smaller AD can represent higher consistency between two priority vectors and further illustrate higher consistency between two domain experts’ judgments. Using this approach to predict quality attributes, if the expression $|NOIV(QA, VS_i) - NOIV(QA, VS_j)| < 0.1$, it is considered that $VS_i$ and $VS_j$ have the same QA level. Then it can be deducted that if AD is less than 0.1, the predicted quality attribute level based on $PV_1$ and the predicted quality attribute level based on $PV_2$ are the same. Therefore, the borderline of AD is 0.1. If the AD is higher than 0.1, the domain experts need to find the biased judgments and modify the comparison matrix until AD is less than 0.1.

The data transfer security (DTSS) in the tourist guide software product line is used as an example to illustrate how to use two domain experts to make judgments in this AHP-based approach. Assume that two domain experts A and B who have rich domain knowledge and experience in the domain of tourist guide software systems perform pair-wise comparisons on the $RF (DTSS, +) = \{Encryption, VirusFilter, WAN, LAN\}$ and $RF (DTSS, -) = \{Web-Based\}$ and generate two comparison matrices $CM_A$ and $CM_B$ as follows:
The CR of CM_A is above 0.1, which means that the pair-wise comparisons made by domain expert A are not consistent. Domain expert A needs to identify the inconsistent pair-wise comparisons and correct them. In this case, three pair-wise comparisons “LAN and Web-Based”, “Web-Based and WAN” and “WAN and LAN” are identified to conflict with each other. Domain expert A makes new judgments and modifies CM_A as follows, and the consistency ratio of the modified comparison matrix is below 0.1.
Once the CR of both \textit{CM}_A and \textit{CM}_B are below 0.1, a priority vector \textit{PV}_A can be calculated from \textit{CM}_A, and a priority vector \textit{PV}_B can be calculated from \textit{CM}_B as follows:

\begin{align*}
\text{Consistency Ratio: 0.044} & \quad \text{Consistency Ratio: 0.021} \\
\text{PV}_A & \quad \text{PV}_B
\end{align*}

Then the average difference between assessed DTSS levels based on \textit{PV}_A and assessed DTSS levels based on \textit{PV}_B is calculated for all the valid selections (\textit{VS}_i) from \textit{RF} (DTSS) as follows:

\[ AD = \sqrt{\frac{1}{n} \sum_{i=1}^{n} \text{NOIV}_1(QA, VS_i) - \text{NOIV}_2(QA, VS_i))^2} = 0.229 \]

The value of \textit{AD} is 0.229 which means two domain experts’ judgments are not consistent. Domain experts \textit{A} and \textit{B} need to negotiate on their judgments. Through negotiation, domain
expert B recognizes that he has biased judgments on the relative importance of feature VirusFilter and modifies CM_B. The modified CM_B and newly calculated PV_B are shown as follows:

```
<table>
<thead>
<tr>
<th>Criteria</th>
<th>Encryption</th>
<th>VirusFilter</th>
<th>WAN</th>
<th>LAN</th>
<th>Web-Based</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encryption</td>
<td>9</td>
<td>6</td>
<td>7</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>VirusFilter</td>
<td>4</td>
<td>2</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WAN</td>
<td>2</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LAN</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Web-Based</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Consistency Ratio: 0.031

Then, the newly calculated AD between PV_A and PV_B is below 0.1, which means that two domain experts’ judgments are consistent. Either PV_A or PV_B can be used as the final priority vector to further establish the interdependency between DTSS and its contributors.

4.3.7 Representing the Interdependencies

As mentioned earlier, the interdependencies between quality attributes and features could be very complex. In industrial-size case studies, a quality attribute (QA) may be affected by hundreds of features while a feature may affect dozens of QA features. The traditional way to represent the feature dependencies, such as requires, excludes or alternative, is not suitable for representing the interdependencies between features and quality attributes. Thus, an xml-based representation schema as illustrated in Figure 4.9 is designed to represent the interdependencies
between a quality attribute and its correlated contributors. Each element and its attributes are described as follows:

```xml
<?xml version="1.0" encoding="GB2312" ?>
  <Interdependencies>
    - <QA name="">
      - <RelevantPositiveFeatures>
        <Feature name="" RIV="" />
      </RelevantPositiveFeatures>
      - <RelevantNegativeFeatures>
        <Feature name="" RIV="" />
      </RelevantNegativeFeatures>
    - <FeatureGroups>
      - <FG type="">
        <Feature name="" />
      </FG>
    </FeatureGroups>
    <VS_OIV max_oiv="" min_oiv="" />
  </QA>
</Interdependencies>
```

**Figure 4.9 The Representation Schema for Interdependency**

- Element `<QA>` represents a quality attribute feature `QA`.
- Element `<RelevantPositiveFeatures>` represent all the contributors that have positive impact on `QA` in `RF (QA)`.
- Element `<RelevantNegativeFeatures>` represent all the contributors that have negative impact on `QA` in `RF (QA)`.
- Element `<Feature>` in `<RelevantPositiveFeatures>` or `<RelevantNegativeFeatures>` represents a contributor of `QA`. Its attribute “name” illustrates the contributor name and attribute “RIV” illustrates the relative importance value of the contributor on `QA`.  

102
- Element `<FeatureGroups>` represents all the feature groups in RF (QA).

- Element `<FG>` under element `<FeatureGroups>` represents a feature group. Its attribute “type” illustrates the type of feature group. Each element `<feature>` under element `<FG>` represents a contributor included in the feature group.

- Element `<VS_OIV>` represents the overall importance value of valid selections from RF (QA). Its attribute “max_oiv” illustrate the maximum OIV and attribute “min_oiv” illustrates the minimum OIV among all the valid selections.

Following this representation schema, the interdependency between quality attribute data transfer speed (DTS) and its contributors can be represented in Figure 4.10.
Once interdependency between a quality attribute feature and its contributors has been measured, the quality level for any feature configuration can be assessed. As described in Section 2.2.1, each feature configuration can be represented as a 2-tuple of the form \((S, R)\) where \(S\) is the set of features to be included and \(R\) is the set of features to be removed \((S \cap R = \emptyset)\). For a feature configuration \(FC = (S, R)\), to assess its relative level on quality attribute \(QA\), a set of steps needs to be followed based on the representation schema of the measured interdependency between \(QA\) and its contributors.

Figure 4.10 The Representation Schema for Data Transfer Speed

```xml
<?xml version="1.0" encoding="GB2312" ?>
<interdependencies>
  <QA name="Data Transfer Speed">
    <RelevantPositiveFeatures>
      <Feature name="Modem19200" RIV="9.87" />
      <Feature name="Modem9600" RIV="6.88" />
      <Feature name="PDA" RIV="3.1" />
      <Feature name="Mobile" RIV="2.76" />
      <Feature name="LAN" RIV="33.37" />
      <Feature name="WAN" RIV="28.56" />
    </RelevantPositiveFeatures>
    <RelevantNegativeFeatures>
      <Feature name="Encryption" RIV="15.45" />
    </RelevantNegativeFeatures>
  </QA>
</interdependencies>
```

Figure 4.10 The Representation Schema for Data Transfer Speed

Once interdependency between a quality attribute feature and its contributors has been measured, the quality level for any feature configuration can be assessed. As described in Section 2.2.1, each feature configuration can be represented as a 2-tuple of the form \((S, R)\) where \(S\) is the set of features to be included and \(R\) is the set of features to be removed \((S \cap R = \emptyset)\). For a feature configuration \(FC = (S, R)\), to assess its relative level on quality attribute \(QA\), a set of steps needs to be followed based on the representation schema of the measured interdependency between \(QA\) and its contributors.
• **Identify Valid Selection.** The first step is to identify the valid selection $VS$ with respect to $QA$ from the feature configuration $FC$ using the formula $VS = (RF (QA, +) \cap S) \cup (RF (QA, -) \cap R)$. The feature set $RF (QA, +)$ and $RF (QA, -)$ and the relative importance value of each feature can be found from the attribute “name” and “$RIV$” of element “Feature” in the representation schema.

• **Calculate OIV for Valid Selection.** The second step is to calculate the overall importance value of the identified valid selection $VS$ on quality attribute $QA$ using the formula $OIV (QA, VS) = \text{Sum} (OIV (QA, f_{g_i} \cap VS))$ where $f_{g_i}$ illustrates one of the feature groups in $RF (QA)$. The feature groups can be found from the element “Feature Groups”. The included features of a feature group $f_{g_i}$ and its type can be found from the attribute “included features” and “type” of the element “$FG$” in the representation schema. To calculate $OIV (QA, f_{g_i} \cap VS)$, it is assumed that $v_{f_{g_i}} = f_{g_i} \cap VS$ and the formulas in Section 4.3.4 are used to calculate the $OIV$s.

• **Normalize OIV into NOIV.** The third step is to calculate the normalized overall importance value ($NOIV$) of the valid selection $VS$ with respect to quality attribute $QA$ using the formula in Section 4.3.5. The expression $\text{MIN} (OIV (VS))$ can be found from the attribute “min_oiv” of the element “$VS_{-OIV}$” while the expression $\text{MAX} (OIV (VS))$ can be found from the attribute “max_oiv” of the element “$VS_{-OIV}$” in the representation schema.
4.3.8 Managing Relationships among Quality Attributes

The measured interdependencies represent the contributing relationship between a quality attribute and its correlated contributors. These interdependencies can be used as reusable assets of the software product line to predict the quality attributes for a configured product. However, if the predicted quality of the configured product cannot satisfy the customers’ quality requirements, the interdependencies cannot be used to find the solutions of modifying the current product to satisfy the customers’ quality requirements. For example, features can be included or removed to achieve the desired QA level based on the interdependency between QA and its contributors. However, it is not clear what impact this modification will have on other quality attributes. In this case, it is found that the information missing in the interdependencies are the relationships among related quality attributes. Without the explicit knowledge of the related quality attributes, it is difficult for application engineers to configure a product that satisfies the customers’ quality requirements in an informed and rational way.

In the context of software product lines, some quality attributes are related with each other as they share one or more contributors. For example, security and performance often exist in a state of mutual tension. A secure system often has the mechanisms to prevent attacks of breaching security. However, the inclusion of security mechanisms into the final system will affect performance negatively, as the responding time will be extended. In this case, security is related with performance as security mechanisms contribute to both of these two quality attributes. This section develops a quality attribute knowledge base (QAKB) to store the information or knowledge about the relationships among related quality attributes in a software
product line. As it is known, the relationships among quality attributes are reflected as the quality attribute levels of software product line members. For example, a product with high security always has low performance. A QAKB is designed as multiple tables, each representing the relationships among a set of related quality attributes. An example of a QAKB table is shown in Table 4.5. The columns of the table represent a set of related quality attributes in a software product line while the rows of the table represent the valid selections with respect to this set of related quality attributes. The corresponding cell \( NOIV (QA_i, VS_j) \) in the table represents the \( QA_i \) level of valid selection \( VS_j \).

<table>
<thead>
<tr>
<th>( VS_j )</th>
<th>( NOIV(QA_1, VS_j) )</th>
<th>( \ldots )</th>
<th>( NOIV(QA_i, VS_j) )</th>
<th>( \ldots )</th>
<th>( NOIV(QA_m, VS_j) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \ldots )</td>
<td>( \ldots )</td>
<td>( \ldots )</td>
<td>( \ldots )</td>
<td>( \ldots )</td>
<td>( \ldots )</td>
</tr>
<tr>
<td>( VS_j )</td>
<td>( NOIV(QA_1, VS_j) )</td>
<td>( \ldots )</td>
<td>( NOIV(QA_i, VS_j) )</td>
<td>( \ldots )</td>
<td>( NOIV(QA_m, VS_j) )</td>
</tr>
<tr>
<td>( \ldots )</td>
<td>( \ldots )</td>
<td>( \ldots )</td>
<td>( \ldots )</td>
<td>( \ldots )</td>
<td>( \ldots )</td>
</tr>
<tr>
<td>( VS_n )</td>
<td>( NOIV(QA_1, VS_n) )</td>
<td>( \ldots )</td>
<td>( NOIV(QA_i, VS_n) )</td>
<td>( \ldots )</td>
<td>( NOIV(QA_m, VS_n) )</td>
</tr>
</tbody>
</table>

To develop a QAKB for a software product line, the quality attributes that are related with each other first need to be identified from the \( QA \) features of the feature model. Two rules are followed to identify the related quality attributes: first, if \( QA_i \) and \( QA_j \) have one or more shared contributors, it can be said that \( QA_i \) is related with \( QA_j \); second, if \( QA_i \) is related with \( QA_j \) and \( QA_j \) is related with \( QA_k \), it can be said that \( QA_i \) is related with \( QA_k \). The \( QAG \) is used to represent a set of related quality attributes identified following the above two rules and in the example of Table 4.5, \( QAG = \{QA_1, QA_2 \ldots QA_m\} \).
To generate a QAKB table for $QAG$ as shown in Table 4.5, two tasks need to be achieved: first, all the valid selections with respect to $QAG$ need to be derived; second, the corresponding $NOIV$ ($QA_i, VS_j$) needs to be calculated. To derive all valid selections with respect to $QAG$, a new feature diagram including the contributors of each quality attribute in $QAG$ is drawn. The selection constraints among the included features can be derived from the original feature diagram. The $RF (QAG)$ is used to represent the contributors of $QAG$ and $RF (QAG) = RF (QA_1) \cup RF (QA_2) \ldots \cup RF (QA_m)$ is defined. From such a feature diagram, a set of valid feature combinations can be derived and these valid feature combinations are also called valid selections ($VSs$) with respect to $QAG$. In the example of Table 4.5, $VSs = \{VS_1, VS_2 \ldots VS_n\}$. As discussed earlier, a valid selection $VS_j$ can be represented as a 2-tuple of the form $(S, R)$ where $S$ represents the set of features included in $VS_j$ among $RF (QAG)$ and $R$ represents the set of features excluded from $VS_j$ among $RF (QAG)$. Then the process in Section 4.3.7 can be used to calculate $NOIV (QA_i, VS_j)$.

In the example of tourist guide software product line of Figure 4.5, a QAKB table can be generated and this QAKB table includes two quality attributes: data transfer speed (DTS) and data transfer security (DTSS) that are related with each other because they share a set of contributors, such as $LAN$, $WAN$ and $Encryption$. A feature diagram, which includes $RF (DTS) \cup RF (DTSS)$ and the constraints among the included features, is generated. 16 valid selections are derived from the feature diagram and a QAKB table is generated as shown in Table 4.6.
Table 4.6 The QAKB Table of Data Transfer Speed and Data Transfer Security

<table>
<thead>
<tr>
<th>Valid Selections for DTS and DTSS</th>
<th>NOIV for DTSS</th>
<th>NOIV for DTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>VS1</td>
<td>WAN, Web-Based, PDA</td>
<td>0.0</td>
</tr>
<tr>
<td>VS2</td>
<td>WAN, Web-Based, PDA, Mobile</td>
<td>0.0</td>
</tr>
<tr>
<td>VS3</td>
<td>WAN, Web-Based, PDA, VirusFilter</td>
<td>0.05</td>
</tr>
<tr>
<td>VS4</td>
<td>WAN, Web-Based, PDA, Mobile, VirusFilter</td>
<td>0.05</td>
</tr>
<tr>
<td>VS5</td>
<td>WAN, Web-Based, PDA, VirusFilter, Encryption</td>
<td>0.74</td>
</tr>
<tr>
<td>VS6</td>
<td>WAN, Web-Based, PDA, Encryption</td>
<td>0.68</td>
</tr>
<tr>
<td>VS7</td>
<td>WAN, Web-Based, Mobile</td>
<td>0.0</td>
</tr>
<tr>
<td>VS8</td>
<td>WAN, Web-Based, Mobile, VirusFilter</td>
<td>0.05</td>
</tr>
<tr>
<td>VS9</td>
<td>LAN, Modem 9600</td>
<td>0.27</td>
</tr>
<tr>
<td>VS10</td>
<td>LAN, Modem 19200</td>
<td>0.27</td>
</tr>
<tr>
<td>VS11</td>
<td>LAN, Modem 9600, Encryption</td>
<td>0.95</td>
</tr>
<tr>
<td>VS12</td>
<td>LAN, Modem 19200, Encryption</td>
<td>0.95</td>
</tr>
<tr>
<td>VS13</td>
<td>LAN, Modem 9600, Encryption, VirusFilter</td>
<td>1.0</td>
</tr>
<tr>
<td>VS14</td>
<td>LAN, Modem 19200, Encryption, VirusFilter</td>
<td>1.0</td>
</tr>
<tr>
<td>VS15</td>
<td>LAN, Modem 9600, VirusFilter</td>
<td>0.32</td>
</tr>
<tr>
<td>VS16</td>
<td>LAN, Modem 19200, VirusFilter</td>
<td>0.32</td>
</tr>
</tbody>
</table>

Once the quality attribute knowledge base for a software product line is generated in domain engineering, this knowledge base can be used to enhance the product configuration with quality information, such as assisting application engineers to derive a product with desired software qualities. The detailed method of using this knowledge base is proposed in next section.

### 4.4 Quality-Aware Product Configuration

The traditional product configuration is the process of selecting the desired features from a feature model based on customers’ functional requirements and the feature constraints specified in a feature model. The customers’ quality requirements are not handled until the final product is
produced. Using QAKB developed by this approach, a quality-aware product configuration is proposed to guide the process of configuring a product from a feature model with a full consideration of the customers’ functional requirements as well as the customers’ quality requirements. As illustrated in Figure 4.11, three main steps need to be followed in the quality-aware product configuration. First, the customers’ requirements are validated to check whether any conflicts exist in the quality requirements. Then, the variable features are included or removed based on the customers’ validated requirements. Finally, the configured product is modified if it fails to satisfy the customers’ quality requirements.

![Figure 4.11 Process of Quality-Aware Product Configuration](image)

To illustrate this process, the following assumptions are given. Assume that a software product line includes \( m \) related quality attributes \( QAG = \{QA_1, QA_2\ldots QA_m\} \). The QAKB table for \( QAG \) is \( qa_kb \) which includes \( n \) valid selections \( VSs = \{vs_1, vs_2\ldots vs_n\} \). The feature model of the software product line is \( fm \) which has \( k \) variation points \( VP_k = \{vp_1, vp_2\ldots vp_k\} \). Among \( VP_k \), \( vp_q \) is a variation point related with implementation techniques and \( vp_q \) corresponds to \( x \) decisions \((d_1, d_2\ldots d_i\ldots d_x)\). In this approach, the \( DQAL (QA) \) in \([0.0, 1.0]\) scale is used to represent the customers’ desired quality level on quality attribute \( QA \) for the target product. Assume that in a product configuration, the customers aim to configure a product with quality requirements
\[ DQAL (QAG) = \{ DQAL (QA_1), DQAL (QA_2) \ldots DQAL (QA_m) \} \]

from the above software product line. The following sub-sections will introduce each individual step of quality-aware product configuration and use the above assumptions to illustrate how to use QAKB to deal with quality related issues in each step.

### 4.4.1 Validating Quality Requirements

In some cases, the customers’ quality requirements cannot be achieved by a software product line, which means that none of the member products can satisfy the customers’ quality requirements. For example, a customer may expect to configure a product with high security, high performance and low purchase cost. Obviously, it is impossible to configure such a product from a software product line as these three quality attributes are related to each other. Before conducting the product configuration, it is necessary to validate the customers’ quality requirements to check whether they can be achieved by the software product line.

The customers’ functional requirements can be validated by checking whether the functional features selected based on the functional requirements satisfy the feature constraints specified in the feature model (Benavides et al. 2005, 2010; Elfaki et al. 2009). This step proposes a method to validate the customers’ quality requirements using the quality attribute knowledge base. The customers can modify their quality requirements based on the validation results.

The method of validating customers’ quality requirements is straightforward. If none of the valid selections in a QAKB table can satisfy the customers’ quality requirements, it can be said
that the customers’ requirements on the set of quality attributes in the QAKB table are invalid.

In the assumed product configuration process, Logical Expression 1 is used to check whether quality requirements $DQAL(QAG)$ are valid through the QAKB table ($qa_kb$). Logical expression 1 means that there exists a valid selection in $qa_kb$ and its quality attributes levels can satisfy the customers’ quality requirements. If this logical expression is true, the customers’ quality requirements are valid; otherwise, the customers’ requirements are invalid. The customers need to modify their desired levels on some quality attributes in $QAG$. The automation of Logical expression 1 is achieved by Algorithm 1.

\[ \exists vs \in VS_s : (\forall qa \in QAG : NOIV(qa, vs) \geq DQAL(qa)) \]

(Logical Expression 1)

---

**Algorithm 1: Validate Customers’ Quality Requirements**

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td><strong>Function</strong> validateQAs (DQAL dqal):: Boolean</td>
</tr>
<tr>
<td>2.</td>
<td><strong>Foreach Configuration</strong> vs in qa_kb. getValidSelections()</td>
</tr>
<tr>
<td>3.</td>
<td><strong>If</strong> (compare (vs, dqal) = = true)</td>
</tr>
<tr>
<td>4.</td>
<td>return true;</td>
</tr>
<tr>
<td>5.</td>
<td><strong>End if</strong></td>
</tr>
<tr>
<td>6.</td>
<td><strong>End foreach</strong></td>
</tr>
<tr>
<td>7.</td>
<td>return false;</td>
</tr>
<tr>
<td>8.</td>
<td><strong>End function</strong></td>
</tr>
</tbody>
</table>

9. **Procedure** compare (Configuration vs, DQAL dqal):: Boolean
   
   // $qa_kb$. getQAs() returns all the related quality attributes in $qa_kb.$

10. **Foreach QA** qa in qa_kb. getQAs()
    
    //vs. get (qa) = NOIV(qa, vs) && dqal. get (qa)= DQAL(qa)
11. \textbf{If} \ ((\text{vs. get}(\text{qa}) < \text{dqal. get}(\text{qa}))) \ \\
12. \quad \text{return} \ false; \ \\
13. \quad \text{End if} \ \\
14. \quad \text{End foreach} \ \\
15. \quad \text{return} \ true; \ \\
16. \quad \text{End procedure} \\

In the example of the tourist guide software product line, suppose that requirement engineers have transferred the customers’ requirements on two quality attributes: data transfer speed (DTS) and data transfer security (DTSS) as follows: \(DQAL\ (DTSS) = 0.5\) and \(DQAL\ (DTS) = 0.5\). From these transferred quality attributes, it is known that the customers expect to configure a product with medium DTSS and medium DTS. It can be found that the desired quality attributes are invalid by using Logical Expression 1 in the QAKB table of Table 4.6 (Section 4.3.8). The customers need to modify their desired levels on these two quality attributes. Assume that the customers modify their quality requirements as \(DQAL\ (DTSS) = 0.3\) and \(DQAL\ (DTS) = 0.4\) which are valid by Logical Expression 1.

\section*{4.4.2 Making Decisions}

After validating the customers’ requirements, the application engineers will make decisions on variation points in the product configuration process. The product configuration can be considered as an iterative decision-making process. In each iterative step, application engineers select a variation point and make a decision on this variation point based on the customers’ requirements. Once a decision has been made, the decision will be propagated to other variation points in the feature model (Zhang et al. 2011). After propagating the decision, a new feature configuration is obtained. If this new feature configuration is still a partial configuration,
another iterative step needs to be performed until a full configuration is obtained.

When making decisions on a specific variation point, there are two scenarios: first, if the variation point is related with the customers’ functional needs, the decision on this variation point will be made based on the customers’ functional requirements. However, if this variation point is related with implementation techniques or operating environments, the customers’ quality requirements can be used as criteria for filtering out the decisions on this variation point, as the implementation techniques or operating environments always have significant impact on software qualities. For example, in the tourist guide software product line, the decision on feature Encryption cannot be made based on the customers’ functional requirements, as the customers have no knowledge about the security mechanism Encryption. The decision on this variation point heavily depends on the customers’ quality requirements. If the customers expect a system with high security, the decision of removing Encryption may be filtered out as the removal of Encryption may lead to a product with an unacceptable security level. If the customers aim to configure a product with low cost, the decision of including Encryption may be filtered out as the inclusion of Encryption may lead to a product exceeding the budget. This step aims to propose a method to filter out the decisions on a variation point based on the customers’ quality requirements by the quality attribute knowledge base. The decisions which will satisfy the customers’ quality requirements on a variation point will be provided to application engineers who will make final decisions.

It is a straightforward task to filter out the decisions on a variation point based on the customers’
quality requirements. If the feature configuration after making a decision on the variation point conflicts with the customers’ quality requirements, this decision will be filtered out. Herein, if a feature configuration conflicts with the customers’ quality requirements, it would mean that starting from this configuration, it would be impossible to derive a product that satisfies the customers’ quality requirements. In the assumed product configuration, when the variation point $vp_q$ is chosen for making decisions, the decisions on $vp_q$ need to be filtered. Assume that the customers have already a partial configuration before making a decision on $vp_q$. As defined earlier, $vp_q$ corresponds to $x$ decisions $(d_1, d_2...d_i...d_x)$ and each decision $d_i$ will result in a new feature configuration $fc_i = \{s_i, r_i\}$. Logical Expression 2 is used to check whether the feature configuration $fc_i$ conflicts with the customers’ quality requirements $DQAL (QAG)$ through QAKB table $qa_kb$. This logical expression means that there exists a valid selection $vs_i$ in $qa_kb$ and it is consistent with $fc_i$ and it satisfies the customers’ quality requirements. If this logical expression is true, the decision $d_i$ will be recommended to application engineers for making decisions; otherwise the decision $d_i$ will be filtered out. Finally, application engineers make final selections from all the recommended decisions. The automation of checking Logical Expression 2 is achieved by Algorithm 2.

$$\exists vs_i \in VS_x : (s_i \subseteq vs_i \land r_i \subseteq \overline{vs_i}) \land (\forall QA_j \in QAG : NOIV (QA_j, vs_i) \geq DQAL(QA_j))$$

(Logical Expression 2)
Algorithm 2: Making Decisions

Global Variables:
qa_kb: the quality attribute knowledge base of software product line

Parameters:
dqal: the customers’ desired levels on quality attributes
con: the current feature configuration.
vp: the current variation point that needs to make decisions.

1. **Function** findValidDecisions (DQAL dqal, Configuration con, VP vp) :: <Collection>

   /// qa_kb.findValidSelections () returns all the valid selections that satisfy dqal in qa_kb.
2. Collection <Configuration> validSelections = qa_kb.findValidSelections(dqal);
3. Collection <Decision> validDecisions = new Collection <> ();
   ///vp.getDecisions () returns the set of possible decisions on variation point vp.
4. Foreach Decision d in vp.getDecisions ()
   ///con.add (d) returns the new feature configuration after making decision d.
5. Configuration new_con = con.add (d);
6. Foreach Configuration vs in validSelections
   /// new_con. consistent (vs) returns true if the valid selection vs is part of configuration new_con,
7. if (new_con. consistent (vs))
8. validDecisions. add (d);
9. End if
10. End foreach
11. End foreach
12. return validDecisions;
13. End function

Following the product configuration in Section 4.4.1, the application engineers could make decisions on a sequence of variation points. Table 4.7 illustrates the process of making decisions on the variation points in the feature model of tourist guide software product line shown in Figure 4.12. As shown in Table 4.7, the decisions on a variation point can be selected based on the customers’ functional requirements in column 3, such as vp1, vp2, vp4 and vp5; otherwise they can be filtered out based on the customers’ quality requirements in column 4, such as vp3. The final decision on any variation point will be propagated to a set of other variation points in
column 5 and column 6. When making decisions on vp3, Logical Expression 2 is used to filter out the decision of including Encryption, as this decision conflicts with the customers’ quality requirements given in Section 4.4.1. Based on this result, application engineers make the decision of removing Encryption. Finally, a configured product is obtained as: \{Tourist Guide, LAN, Position Detection, Satellite, Modem19200, Route Search, Modem\} \{WAN, Web-Based, Terminal Device, PDA, Mobile, Encryption, Modem 9600, VirusFilter\}.

<table>
<thead>
<tr>
<th>Step</th>
<th>Variation Point</th>
<th>Decision</th>
<th>Feature relationship propagation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>VP1</td>
<td>Include LAN</td>
<td>Modem WAN, Web-Based, Terminal Device, Mobile, PDA</td>
</tr>
<tr>
<td>2</td>
<td>VP2</td>
<td>Include Position Detection</td>
<td>Satellite</td>
</tr>
<tr>
<td>3</td>
<td>VP3</td>
<td>Remove Encryption</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>VP4</td>
<td>Include Modem19200</td>
<td>Modem9600</td>
</tr>
<tr>
<td>5</td>
<td>VP5</td>
<td>Remove VirusFilter</td>
<td></td>
</tr>
</tbody>
</table>
### 4.4.3 Modifying Configured Products

Before using the configured product to guide further software development, it is necessary to check whether the predicted quality attributes of the configured product can satisfy the customers’ quality requirements. The quality attributes of a configured product can be predicted by the quality attribute knowledge base. If the predicted quality attributes can satisfy the customers’ quality requirements, this configured product can be used as the final product for further software development; otherwise, it must be modified to achieve the desired software qualities. In the latter case, the customers themselves have no knowledge about how to modify the existing configured product to achieve their quality requirements. To assist the customers to modify the configured product, a set of solutions can be recommended and the customers can choose their desired one with a full consideration of the changed functionalities and the changed quality attributes. This step aims to propose a method to find all the solutions of modifying the existing configured product to satisfy the customers’ quality requirements by a quality attribute.
knowledge base. Then application engineers can select the desired solution with a full consideration of the recommended solutions.

The task of finding the modification solutions can be achieved by finding all the valid selections that satisfy the customers’ quality requirements in the QAKB table, and then comparing these valid selections with the existing configured product. In the assumed product configuration process, suppose that a configured product $pc = \{s, r\}$ is derived from the feature model. The valid selection of $pc$ for $QAG$ can be identified as $vs = VS (QAG, pc)$. The predicted quality attributes of $pc$ on $QAG$ can be obtained from $qa_kb$ through $vs$ and they can be represented as $NOIV (QA_1, vs), \ldots NOIV (QA_m, vs)$. If the predicted quality attributes of $pc$ cannot satisfy the customers’ quality requirements, $pc$ needs to be modified to achieve the desired software qualities. In $qa_kb$, each valid selection which satisfies the customers’ quality requirements can indicate a modification solution. This modification solution can be obtained by comparing the existing configured product with this valid selection. $Include (set)$ is used to represent the set of features that need to be newly included into $pc$ and $Remove (set)$ is used to represent the set of features that need to be removed from $pc$ in a modified solution corresponding to valid selection $vs$, which satisfies the customers’ quality requirements. The feature set $Include (set)$ and $Remove (set)$ can be obtained by assigning Logical Expression 3 and Logical Expression 4 as “true” respectively. After obtaining all the modified solutions, the customers can specify extra criteria to reduce the number of solutions. Finally, the customers can select their desired solution based on $Include (set), Remove (set)$ and the quality attribute levels of the modified product. The automation of Logical Expression 3 and 4 is realized in
Algorithm 3: Finding Modification Solutions

Global Variables:
qa_kb: the quality attribute knowledge base of software product line

Parameters:
dqal: the customers’ desired levels on quality attributes
con: the final configuration.

1. **Function** findSolutions (DQAL dqal): Collection<Solution>
2. Collection <Solution> solutions = new Set <Solution> ();
3. Collection <Configuration> validSelections= qa_kb. findValidSelections (dqal);
4. **Foreach** Configuration vs in validSelections
5. Solution s = new Solution ();
   // getAllFeatures() return all the features in the feature model
6. **Foreach** Feature f in getAllFeatures()
   // con. includes (f) returns true if f is included in the configuration con and con. excludes (f) returns true if f is excluded from con.
7. **If** (vs. includes (f) && con. excludes(f))
   s. addIncludedFeatures (f);
8. **End If**
9. **If** (vs. excludes (f) && con. includes(f))
10. s. addRemovedFeatures (f);
11. **End If**
12. **End foreach**
13. **End function**
Following the product configuration process in Section 4.4.2, the quality attribute levels of the configured product can be predicted using the QAKB table of Table 4.6. The predicted results are \( NOIV(DTSS, PC) = 0.27 \) and \( NOIV(DTS, PC) = 1.0 \). Comparing with the customers’ quality requirements \( DQAL(DTSS) = 0.3 \) and \( DQAL(DTS) = 0.4 \), it is found that this configured product needs to be modified to increase the DTSS level. By using the Logical Expression 3 and 4, four modification solutions can be found as listed in Table 4.8. Application engineers can choose the best solution with a full consideration of changed functional features and the quality attributes levels.

<table>
<thead>
<tr>
<th>Solution</th>
<th>Inclusion</th>
<th>Removal</th>
<th>DTS</th>
<th>DTSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Encryption</td>
<td></td>
<td>0.95</td>
<td>0.43</td>
</tr>
<tr>
<td>2</td>
<td>Encryption, VirusFilter</td>
<td></td>
<td>1.0</td>
<td>0.43</td>
</tr>
<tr>
<td>3</td>
<td>VirusFilter</td>
<td></td>
<td>0.32</td>
<td>1.0</td>
</tr>
<tr>
<td>4</td>
<td>VirusFilter, Modem19200</td>
<td>Modem9600</td>
<td>0.32</td>
<td>0.89</td>
</tr>
</tbody>
</table>

### 4.5 Concluding Remarks

This chapter proposes an approach of modeling quality attributes in a feature model and conducting quality-aware product configuration. The analytic hierarchical process (AHP) is used to assist domain experts in making judgments. This proposed approach has several advantages compared with the existing approaches in literature. First, this approach is more efficient than the approaches based on real products, because it is difficult and time-consuming
to produce real products in practice. Second, this approach involves less human efforts than other approaches based on domain experts’ judgments, such as BBN approach (Zhang et al. 2003). For \( n \) relevant features of a quality attribute, the number of judgments required by the AHP method in this approach is \( n(n-1)/2 \) which is much less than \( 2^n \) required by the BBN approach. Also, it is much easier for a domain expert to compare the relative importance of a pair of features in this approach than to judge the overall impact of a combination of features in the BBN approach.

This quality modeling approach can be considered as a starting point for the quality assessment through all the stages of product derivation. As discussed earlier, the product quality can be affected by all the lifecycles of product development in a software product line, such as product configuration, software architecture derivation and reusable components composition. The approach proposed in this chapter focuses on assessing the quality levels for a product configured from feature models, but not for the final generated product. More activities for quality assessment can be carried out in the later stages of product development.
5. Prototype Tool

This chapter introduces the prototype tool which is developed to implement the concepts of the approaches proposed in this thesis. This prototype tool includes four main parts: one tool named as feature model editor tool (FMETool) aims to edit feature models, one tool named as feature model validation tool (FMVTool) aims to validate a feature model; one tool named as quality attributes modelling tool (QAMTool) aims to model quality attributes in a feature model and one tool named as quality aware product configuration tool (QAPCTool) aims to support product configuration based on feature models.

5.1 Feature Model Editor Tool

The FMETool is used to build and edit feature models. FMETool provides two ways of building a feature model: first, the feature modelers can use FMETool to construct a feature model. As shown in Figure 5.1, a root feature for a feature model can be first created and then sub-features under a certain feature are added following different hierarchical relationships, including mandatory, optional, variation point and variant. The feature name and description can be set when creating a feature. The cardinality can be set for a variation point feature. The cross-tree constraints, requires and excludes, can be added between two features.
FMETool also provides the function of generating random feature models. In literature, a set of tools has been developed to generate random feature models as test cases (Segura 2008; Segura et al. 2012; Yan et al. 2009). To better integrate the function of generating random feature models with the rest of tools, the idea in BeTTy (Segura et al. 2012) is adapted by FMETool in the following steps. A feature tree is first generated. Starting with a single root feature, it runs several iterations. In each iteration step, an existing feature without child features is randomly selected and one to \( n \) (random number) child features are added to it. The child features are connected with their parent by mandatory or variation point following a defined probability. The iteration process terminates until the feature tree has enough features. After a feature tree is generated, the cross-tree constraints are added following three rules: a constraint cannot exist between two features with the parental relationships; two features cannot share more than one
constraint; and both of two features cannot be full-mandatory features. As shown in Figure 5.2, a set of parameters need to be set when generating random feature models, such as the number of features, the number of cross-tree constraints, the percentage of different types of features, the maximum height of feature tree, the maximum number of child features under a certain feature and the maximum variant features under a variation point.

![Figure 5.2 Function View of Generating Feature Models in FMETool](image)

Once a feature model is constructed or randomly generated by FMETool, it can be exported into an xml file as shown in Figure 5.3. This format is the same as the one used by feature model analysis tool (FAMA) (Benavides et al. 2007). The `<binaryRelation>` with `<cardinality min = “0”, max = “1”>` represents optional relationships and the `<binaryRelation>` with `<cardinality min = “1”, max = “1”>` represents mandatory relationships. The `<solitaryFeature>` presents a
child feature that connects with its parent feature by optional or mandatory relationship. The
<setRelation> with <cardinality min = “a”, max = “b”> represents a variation point with
cardinality (a…b). Each variant feature under a variation point is represented by
<groupedFeature> under <setRelation>. The “requires” and “excludes” constraints are
described by the elements <requires> and <excludes> respectively.

```xml
<?xml version="1.0" encoding="GB2312" ?>
<feature-model>
  <feature name="">
    <binaryRelation name="">
      <cardinality min="" max="" />
    </binaryRelation>
    <setRelation name="">
      <cardinality min="" max="" />
      <groupedFeature name="" />
      <groupedFeature name="" />
    </setRelation>
  </feature>
<qaFeature name="" />
<requires name="" feature="" requires="" />
<excludes name="" feature="" excludes="" />
</feature-model>
```

Figure 5.3 XML Format for Representing Feature Models

5.2 Feature Model Validation Tool

The FMVTool is developed based on the validation approach proposed in Chapter 3. The main
goal of the FMVTool is to identify the errors for a feature model and provide the explanations
for a specific error. Then the feature modelers can resolve the errors based on the validation
results. As shown in Figure 5.4, an existing feature model is first loaded into FMVTool from an
xml file. Then the dead features and false variable features in the loaded feature model can be
found by pressing “Find All Dead” and “Find All False” respectively. All the identified feature model errors will appear in the listbox of “Feature Model Errors”. To explain an error on a specific feature \( f \), \( f \) is selected in the listbox of “Feature Model Errors” and “Explain” is pressed. Then all the contradictory relationship sets that cause the error and the minimal explanations of the error will be shown in the box of “Error Explanation”. Based on the error explanations, the feature modelers can resolve the errors by modifying the feature relationships in two ways: modifying the xml file which represents the feature model or editing the feature model from the function view of Figure 5.2. Figure 5.4 shows the process of using FMVTool to validate the feature model of the tourist guide software product line of Figure 2.2. The validation results show that Mobile is identified as a dead feature and this error is caused by the contradictory relationship set \{Rq-8, Ex-5, SR\}. Three minimal explanations \{Rq-8\}, \{Ex-5\} and \{SR\} are returned from FMVTool.
5.3 Quality Attributes Modeling Tool

The QAMTool is developed based on the quality attributes modeling approach proposed in Chapter 4. The main goal of the QAMTool is to assist domain engineers measure the interdependencies between features and quality attributes based on domain experts’ judgments. The input of QAMTool is a valid feature model while the output of QAMTool is a set of measured interdependencies. To illustrate how to use QAMTool, the valid feature model of the tourist guide SPL is used as an example. As shown in Figure 5.5, a feature model is first loaded into QAMTool. The feature diagram and QA feature diagram are shown separately.
Once a valid feature model is loaded into QAMTool, the interdependencies between features and quality attributes are measured in four steps. The first step is to identify the contributors of quality attributes. As shown in Figure 5.6, a quality attribute (QA) for measurement is first chosen by selecting QA from the list box of “Identify Contributors” and “Confirm QA Feature” is pressed. In order to confirm a positive contributor for QA, the feature in the feature diagram is selected and then “Positive” is pressed. Then the confirmed contributor will appear in the list box of “Positive Contributors”. Similarly, to confirm a negative contributor, the feature in the feature diagram is selected and “Negative” is pressed. Then the contributor will appear in the list box of “Negative Contributors”. The result of this step is a quality attribute with its related contributors and can be exported into an xml file by pressing “Export”. Figure 5.6 shows the
function view of confirming the contributors for the quality attribute data transfer speed in the
tourist guide SPL.

Figure 5.6 Function View of Identifying Contributors in QAMTool

The second step of using QAMTool is to prioritize the identified contributors of QA. As shown
in Figure 5.7, the contributors of QA are loaded into QAMTool and shown as the rows and
columns of the “Comparison Matrix”. Then the pair-wise comparisons among the contributors
are conducted and each comparison is assigned with a value in [-9.0, +9.0] scale, based on the
domain experts’ judgments. Once the matrix is finalized, the relative importance values (RIV) of
the contributors can be calculated by pressing “Calculate RIV”. The result will be shown in the
list box of “RIV of Contributors”. The method used to calculate the RIVs is to calculate the
normalized Eigen vector of the comparison matrix (Hallowell and David 2007). Figure 5.7
shows the result of calculating the RIVs of the contributors of data transfer speed in the tourist guide SPL.

![Figure 5.7 Function View of Prioritizing Contributors in QAMTool](image)

The third step of using QAMTool is to identify the feature groups among the contributors of QA. The current version of QAMTool provides four types of feature groups: \textit{SumGp}, \textit{AvgGp}, \textit{MinGp} and \textit{MaxGp} and domain-specific feature groups can also be defined. The final step of using QAMTool is to calculate the overall importance values (\textit{OIV}) of all valid products and find the maximum \textit{OIV} and the minimum \textit{OIV}. FAMA (Benavides et al. 2007) is used to find all valid products. Following the above four steps, the interdependency between a quality attribute QA and its contributors can be generated from QAMTool. The interdependency include the contributors of QA, their relative importance values (RIV), the feature groups that the
contributors belong to, and the maximum and minimum $OIV$. The measured interdependency will be stored into an xml file which will be used in product configuration.

### 5.4 Product Configuration Tool

The QAPCTool is developed based on the approach proposed in Chapter 4. The main goal of QAPCTool is to guide the application engineers to configure a desired product in a quality-aware manner. The input of QAPCTool is a valid feature model and a set of measured interdependencies while the output of QAPCTool is a valid configured product.

The first step of using QAPCTool is to validate the customers’ quality requirements. As shown in Figure 5.8, application engineers first interpret the customers’ quality requirements into a number in [0.0, 1.0] scale and load the quality requirements into QAPCTool by pressing “Configure QA”. Then the loaded requirements can be checked for conflicts by pressing “Validate Quality Requirements”. If it is found that no valid products can satisfy the loaded quality requirements, QAPCTool will return “Invalid Quality Requirements” to indicate that the customers need to modify their requirements on the quality attributes. Figure 5.8 shows that the loaded quality requirements in the list box “Configured Quality Attributes” have conflicts and that the application engineers modify the requirements as 0.4 for data transfer speed and 0.4 for data transfer security by negotiating with the customers.
After validating the customers’ quality requirements, the second step of using QAPCTool is to select or remove features. As shown in Figure 5.9, the inclusion or removal of a feature is achieved by selecting the feature in the feature diagram and pressing “Select” or “Remove” respectively. As the decision on a feature will be propagated to other features automatically based on the propagation algorithm proposed in (Zhang et al. 2011), inconsistent decisions can be avoided. If a decision conflicts with the customers’ quality requirements, the alternative solutions will be provided in the list box of “Decision Filter”. If a wrong decision is made, QAPCTool has the “Rollback” function which can recover all the features affected by the wrong decision. Figure 5.9 shows the selection of LAN leads to the inclusion of Modem and the removal of WAN, Web-Based, Terminal Device, Mobile and PDA.
The third step of using QAPCTool is to modify a configured product if its predicted quality attributes cannot satisfy the customers’ requirements. As shown in Figure 5.10, QAPCTool can provide a set of solutions in the list box of “Modification Solutions”. Each solution includes the set of features that need to be included, the set of features that need to be removed, and quality attribute levels of the modified product. A modification solution shown in Figure 5.10 is removing Encryption and VirusFilter from the current product and the quality levels of the modified product is 0.43 for data transfer speed and 1.0 for data transfer security. Finally, the desired solution can be chosen with a full consideration of the changed functionalities and the changed quality attributes and the final product is exported into an xml file.
This chapter introduces a prototype tool that implements the approaches proposed in this research. This tool supports a set of activities from domain engineering to application engineering, such as feature modeling, feature model validation, quality attributes modeling in a feature model and quality-aware product configuration. Currently, this tool is publicly available and interested readers can download the tool suite from the web link: spl.uon.apptao.com. This tool enhances the applicability of the framework proposed in this thesis.
6. Evaluation

This chapter records a set of empirical experiments which were carried out to evaluate the approaches proposed in this thesis. In particular, my goal was to evaluate the validation approach proposed in Chapter 3 in terms of the correctness of identified errors and the validation efficiency. In addition, we aimed to evaluate the quality assessment approach proposed in Chapter 4 in terms of the precision of the assessed product qualities. We will first explain how the experiments were designed and introduce the resources used in the experiments. Then we will report and discuss the observed results.

6.1 Evaluation of Feature Model Validation

This section evaluates the proposed validation approach from two aspects: the correctness of the identified feature model errors and their explanations and the performance for debugging large-scale feature models. An experiment as shown in Figure 6.1 was designed for the evaluation.

For a specific feature model, a well-known constraint satisfaction problem based approach (Trinidad et al. 2007) and its supporting tool feature model analysis tool (FAMA) (Benavides et al. 2007) are used to identify feature model errors and find the explanations for the identified errors. Meanwhile, the validation approach proposed in Chapter 3 and its corresponding prototype tool FMVTool introduced in Section 5.2 is used to validate the same feature model. The correctness of the proposed approach can be verified by comparing the validation results returned from FMVTool and FAMA. The efficiency of the proposed approach can be evaluated through comparing the time spent by FAMA and FMVTool for validating the same feature models.
The correctness of the proposed validation approach heavily relies on whether the identified feature model errors and their explanations are complete and correct. To evaluate the correctness of the proposed approach, FMVTool and FAMA are used to validate a set of pre-designed feature models as shown in Table 6.1 and the validation results are compared. All kinds of feature model errors are included into these examples, such as dead features and false variable features. From another aspect, these examples include feature model errors caused by one contradictory relationship set (CRS), by multiple CRSs that have no interactions and by multiple CRSs that have intersections. The validation results show that the proposed validation approach and the prototype tool FMVTool can identify feature model errors correctly and provide the right error explanations on these examples. It should be noted that the explanations identified by FMVTool can cover the explanations identified by FAMA. This is because that the proposed validation approach finds the minimal explanations as introduced in Section 3.4, whereas FAMA finds the minimum explanation which is an explanation that contains the minimum set of failing feature relationships while giving the reason of the error (Trinidad et al.)
2008). For example, in the third feature model of Table 6.1, FAMA can identify the minimum explanation \( \{Rq-1\} \) for dead feature A and FMVTool can identify three minimal explanations \( \{Rq-1\}, \{Ex-1, Rq-2\} \) and \( \{Ex-1, Ex-2\} \) for dead feature A.

**Table 6.1 Validation Results from FAMA and FMVTool on Pre-designed Feature Models**

<table>
<thead>
<tr>
<th>Feature Models</th>
<th>Validation Result from FAMA</th>
<th>Validation Result from FMVTool</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dead Feature: C3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Minimum</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Explanations:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SR-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ex-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rq-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>False-mandatory Feature: D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Minimum</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Explanations:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SR-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rq-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BR-2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ex-1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The next step is to validate the proposed validation approach based on some randomly generated feature models. The numbers are used to name the features in the generated feature models. For example, if a feature model with five features is generated, the names of the five features in the feature model are 1, 2, 3, 4, and 5. Fifteen feature models are randomly generated as shown in Table 6.2. In these feature models, the number of features ranges from 20 to 40 and the number of feature constraints ranges from 10 to 20. Table 6.2 shows the results of using the validation tool FMVTool to identify dead features and false variable features from these generated feature models. By comparing the validation results in Table 6.2 with the validation results returned from FAMA, it can be concluded that FMVTool and FAMA can identify the
same set of feature model errors for these randomly generated feature models. Furthermore, it is found that the minimal explanations identified by FMVTool can equal with or cover the minimum explanations provided by FAMA on these generated feature models. For example, Figure 6.2 shows the random feature model 2 in Table 6.2. Using FMVTool, the minimal explanations of dead feature 13 can be identified as \{Rq-2\}, \{BR-7\}, \{BR-8\}, \{Rq-1\} and \{Ex-1\}; and the minimal explanations of false variable feature 2 can be identified as \{Rq-3\}, \{SR-4\}, \{Rq-4\} and \{Rq-5\}. This result is the same as the validation result returned from FAMA. Therefore, it can be concluded that the proposed validation approach can identify and explain feature model errors correctly on these randomly generated feature models.

<table>
<thead>
<tr>
<th>Feature Model</th>
<th>Num of Features</th>
<th>Num of Requires</th>
<th>Num of Excludes</th>
<th>Dead Feature Error</th>
<th>False Variable Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20</td>
<td>5</td>
<td>5</td>
<td>5, 18, 11, 12, 17</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td>5</td>
<td>5</td>
<td>13</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td>5</td>
<td>5</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>30</td>
<td>10</td>
<td>10</td>
<td>18, 19, 27</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>30</td>
<td>10</td>
<td>10</td>
<td>20, 25, 30</td>
<td>10, 19</td>
</tr>
<tr>
<td>6</td>
<td>30</td>
<td>10</td>
<td>10</td>
<td>4, 15, 16, 17, 22, 25</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>30</td>
<td>10</td>
<td>10</td>
<td>9, 25, 26, 27, 28, 29, 30, 15, 2, 7, 23, 24, 8</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>30</td>
<td>10</td>
<td>10</td>
<td>21, 7, 24, 4, 16, 17</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>30</td>
<td>10</td>
<td>10</td>
<td>21, 7, 24, 4, 16, 17</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>30</td>
<td>10</td>
<td>10</td>
<td>5, 24, 25, 28, 16</td>
<td>23</td>
</tr>
<tr>
<td>11</td>
<td>40</td>
<td>10</td>
<td>10</td>
<td>5, 24, 25, 28, 16</td>
<td>23</td>
</tr>
<tr>
<td>12</td>
<td>40</td>
<td>10</td>
<td>10</td>
<td>37, 18, 20, 21, 22, 24, 25, 27</td>
<td>26</td>
</tr>
<tr>
<td>13</td>
<td>40</td>
<td>10</td>
<td>10</td>
<td>30, 9, 34, 12, 35, 36, 37, 38</td>
<td></td>
</tr>
</tbody>
</table>
Figure 6.2 An Example of Random Feature Model

The performance of the proposed validation approach can be verified by the following experiments. Five groups of feature models are randomly generated and each group consists of 10 feature models. The feature models in the same group have the same number of features and the same number of cross-tree constraints. Then FMVTool and FAMA are used to validate five groups of randomly generated feature models. The average time spent by FAMA and the average time spent by FMVTool for validating each group of feature models is recorded in Table 6.3. The validation results show that FAMA reveals a weak time performance when resolving feature model errors on medium and large-scale feature models. For validating a feature model which has more than 100 features, FAMA even meets “out of memory” error due to the state space explosion problem. Compared with FAMA, FMVTool reveals a strong time performance and finds the explanations for the feature model errors in seconds.
Table 6.3 Time Spent for Identifying Feature Model Errors by FMVTool and FAMA

<table>
<thead>
<tr>
<th>Feature Model Group</th>
<th>Features</th>
<th>Requires</th>
<th>Excludes</th>
<th>FMVTool (ms)</th>
<th>FAMA (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20</td>
<td>5</td>
<td>5</td>
<td>100</td>
<td>6968</td>
</tr>
<tr>
<td>2</td>
<td>30</td>
<td>10</td>
<td>10</td>
<td>160</td>
<td>10200</td>
</tr>
<tr>
<td>3</td>
<td>40</td>
<td>10</td>
<td>10</td>
<td>320</td>
<td>15600</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>10</td>
<td>10</td>
<td>610</td>
<td>Error</td>
</tr>
<tr>
<td>5</td>
<td>1000</td>
<td>50</td>
<td>50</td>
<td>1180</td>
<td>Error</td>
</tr>
</tbody>
</table>

In conclusion, it is observed that the validation tool built by this research work can produce the same results as FAMA for the same test cases (a set of pre-designed feature models and randomly generated feature models). This observation increases the level of confidence on the correctness of this proposed validation approach. For the validation performance, it is found that the built validation tool never fails on the test cases and only needs a few seconds to generate the validation results for large-scale feature models (more than 100 features).

### 6.2 Evaluation of Quality Assessment

This section evaluates the quality assessment approach proposed in Chapter 4. The usefulness of this approach heavily relies on the precision of the quality levels predicted for a configured product. An experiment was designed as shown in Figure 6.3. For a configured product, its quality levels are predicted by the proposed quality assessment approach. Meanwhile, the quality levels of the configured product are estimated by domain experts who have domain experience and knowledge. The precision of the assessed product quality can be evaluated by comparing the quality levels predicted by the proposed approach and the quality levels estimated by
domain experts. It should be noted that the domain experts who are used for estimation must be different from those who are used for judgments in the analytic hierarchical process (AHP) of the proposed approach, as it is not rational for a domain expert to assess his own judgments. In this experiment, the domain experts used for judgments in AHP are named as judging domain experts while the domain experts used for estimation are named as testing domain experts. The domain experts who were used in this experiment were Ph.D students with software engineering background and were trained to understand the domain knowledge.

The case study used for evaluation is a computer-aided dispatch (CAD) software product line.

The primary mission of a mission-critical CAD system is to respond to emergency events, such as hazards associated with industry, crime and natural risks (wildfires, earthquakes, flooding and severe weather). Figure 6.4 illustrates the basic functionalities and roles of a CAD system. When a “caller” reports an incident by phone call, a “call taker” in the control center obtains the details about the incident and creates a task for the incident. The new task will be added to a list of un-dispatched tasks. The “dispatcher” examines the situation, selects the most appropriate
and available “resources” and dispatches them to execute the task. The “resources” carry out the task instructions and report to the “task manager” who actively monitors the situation and closes the task when it is completed.

Currently, there are several CAD software providers in the market, such as EnRoute, InterAct, Intergraph, Positron etc. They provide a wide array of CAD software systems ranging from homeland security, fire departments and emergency medical teams to agency personnel management. The need to develop and customize specific CAD software systems for specific companies or organizations motivates the CAD software product line. By studying and analyzing a set of existing CAD software systems, their commonalities and variabilities have been identified and a feature model for CAD software product line is established as shown in Figure 6.5 and Figure 6.6.
Figure 6.5 Feature Model of CAD Software Product Line (1)
Figure 6.6 Feature Model of CAD Software Product Line (2)
Starting from the feature model of CAD software product line, the quality attributes modeling approach proposed in Chapter 4 is used to identify and represent quality attributes in a sub-tree of a feature model as illustrated in Figure 6.7. Performance, security, usability and low cost are identified as the abstract quality attributes of CAD software product line. The low cost is further specialized into purchase low cost and maintenance low cost. The security is decomposed into data transfer security and data access security. The performance is specialized into dispatch respond time, availability, data transfer speed and utilization of resources. The usability is specialized into easier to learn, efficient to use and ease of use.

![Figure 6.7 Quality Attribute Feature Tree of CAD Software Product Line](image)

In the QA feature diagram of Figure 6.7, six QA features are chosen for interdependency measurement: data access security (DAS), dispatch respond time (DRT), ease of use (EOU), maintenance low cost (MLC), data transfer speed (DTS) and data transfer security (DTSS). Two domain experts (A and B) are used to measure the interdependencies or estimate the product
quality in two phases of the designed experiment. In the first phase, domain expert A measures the interdependencies for EOC, DAS and DRT while domain expert B estimates the quality levels for the configured products on these three QA features. In the second phase, domain expert B measures the interdependencies for MLC, DTS and DTSS while domain expert A estimates the quality levels for the configured products. In the first phase, the interdependencies are measured by domain expert A following the steps in Section 4.3. The measured results are shown in Figure 6.8, Figure 6.9 and Figure 6.10.

```xml
<?xml version='1.0' encoding="GB2312" ?>
<Interdependencies>
  <QA name="Ease of Use">
    <RelevantPositiveFeatures>
      <Feature name="Command-Line" RIV="4.66"/>
      <Feature name="Drag-Drop" RIV="51.17"/>
      <Feature name="Colour-Based" RIV="22.79"/>
      <Feature name="Timer Alerts" RIV="6.75"/>
      <Feature name="One-Login" RIV="14.63"/>
    </RelevantPositiveFeatures>
    <FeatureGroups>
      <FG type="MaxGp">
        <Feature name="Drag-Drop"/>
        <Feature name="Command-Line"/>
      </FG>
      <FG type="SumGp">
        <Feature name="Colour-Based"/>
        <Feature name="Timer Alerts"/>
        <Feature name="One-Login"/>
      </FG>
    </FeatureGroups>
    <VS_OIV max_oiv="95.34" min_oiv="4.66"/>
  </QA>
</Interdependencies>
```

Figure 6.8 Interdependency of Ease of Use
Figure 6.9 Interdependency of Data Access Security

Figure 6.10 Interdependency of Dispatch Respond Time
Then FAMA (Benavides et al. 2007) is used to generate 10 random products from the CAD feature model and list them in Table 6.5. It should be noted that only the features that contribute to one of the three quality attributes DAS, EOU, and DRT are considered. The features that contribute to one of the three quality attributes are listed in Table 6.4.

<table>
<thead>
<tr>
<th>Table 6.4 Contributors of DAS, EOU and DRT</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Features</strong></td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 6.5 Configured Products from Feature Model of CAD Software Product Line</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Selected Features</strong></td>
</tr>
<tr>
<td>FC1</td>
</tr>
<tr>
<td>FC2</td>
</tr>
<tr>
<td>FC3</td>
</tr>
<tr>
<td>FC4</td>
</tr>
<tr>
<td>FC5</td>
</tr>
<tr>
<td>FC6</td>
</tr>
<tr>
<td>FC7</td>
</tr>
<tr>
<td>FC8</td>
</tr>
<tr>
<td>FC9</td>
</tr>
<tr>
<td>FC10</td>
</tr>
</tbody>
</table>
For each configured product, its quality levels are assessed based on the interdependencies measured by domain expert A as well as estimated by domain expert B. The quality levels predicted by the proposed approach are represented in $[0.0, 1.0]$ scale. The value “1” illustrates the highest quality level while the value “0” illustrates the lowest quality level. Domain expert B estimates the product quality in three qualitative levels: low, medium and high. To compare the results, it is necessary to transform the predicted quality attributes in $[0.0, 1.0]$ scale into three qualitative levels. In this experiment, a value in $[0.0, 0.35]$ can be considered as low level, a value in $[0.35, 0.7]$ can be considered as medium level while a value in $[0.7, 1.0]$ can be considered as high level. The comparison results are shown in Table 6.6 and the pairs that are not matched are italicized and capitalized.

<table>
<thead>
<tr>
<th></th>
<th>$DAS$</th>
<th>$EOU$</th>
<th>$DRT$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AHP-based Approach</td>
<td>Testing domain experts</td>
<td>AHP-based Approach</td>
</tr>
<tr>
<td>$FC1$</td>
<td>0.583</td>
<td>medium</td>
<td>0.512</td>
</tr>
<tr>
<td>$FC2$</td>
<td>0.138</td>
<td>MEDIUM</td>
<td>0.587</td>
</tr>
<tr>
<td>$FC3$</td>
<td>0.70</td>
<td>high</td>
<td>0.764</td>
</tr>
<tr>
<td>$FC4$</td>
<td>0.0</td>
<td>low</td>
<td>0.674</td>
</tr>
<tr>
<td>$FC5$</td>
<td>1.0</td>
<td>high</td>
<td>0.83</td>
</tr>
<tr>
<td>$FC6$</td>
<td>0.36</td>
<td>medium</td>
<td>0.83</td>
</tr>
<tr>
<td>$FC7$</td>
<td>0.22</td>
<td>low</td>
<td>0.251</td>
</tr>
<tr>
<td>$FC8$</td>
<td>0.78</td>
<td>high</td>
<td>0.235</td>
</tr>
</tbody>
</table>
In the second phase of the designed experiment, domain expert B measures the interdependencies for the other three QA features: MLC, DTS and DTSS as shown in Figure 6.11. The quality levels of the 10 configured products in Table 6.5 are predicted based on the interdependencies of Figure 6.11 as well as estimated by domain expert A. Then the predicted quality levels are compared with the estimated quality levels.

Figure 6.11 Interdependency of MLC, DTS and DTSS

A set of lessons can be learnt from the evaluation:
• From the evaluation result, it is found that 83% of quality attributes predicted by the proposed quality assessment approach match the quality attributes estimated by domain experts. All six mismatches occur between adjacent levels, i.e., low level predicted by the proposed approach is estimated as medium level, medium level predicted by the proposed approach is estimated as low or high level, or high level predicted by the proposed approach is estimated as medium level. Based on the above result, it can be said that the quality attributes predicted by the proposed approach are precise to a great extent and the usefulness of the proposed approach can be verified by the evaluation.

• The size of a quality attribute tree is not very large in the real case studies. This fact can be explained from two aspects. First, the proposed approach only deals with the feature-sensitive quality attributes which will be affected by the selection or removal of features. The quality attributes which are sensitive to architecture designs and implementation techniques are not considered by the proposed approach or included in the quality attribute tree. Second, only the quality attributes that are critical to the software product line domain are kept and represented in the quality attribute feature tree.

• The domain experts’ efforts involved in the judgments are not very onerous. In the above experiment, a domain expert makes 10 pair-wise comparisons for EOU, 15 pair-wise comparisons for DAS and 15 pair-wise comparisons for DRT. Each pair-wise comparison requires 30 seconds on average and 45 minutes are needed for measuring the interdependencies in the above case study. In the contrast, the BBN approach (Zhang et
al. 2003) needs 160 judgments on the same case study and each judgment requires at least 2 minutes on average. From this experiment, it is found that the efforts involved in the AHP method are not very onerous; despite the AHP method requiring a large number of judgments. This is mainly because it is quite easy for a domain expert to make judgments on the relative importance of a pair of features and it only takes a short time (less than one minute) to make a pair-wise comparison. Therefore, the domain experts’ efforts required by the proposed approach are significantly reduced.

Furthermore, a set of problems are identified as the major obstacles of applying the proposed approach in practice:

- The number of judgments made in the AHP method is \( n(n-1)/2 \). As two domain experts are needed to make judgments in the AHP-based approach, the number of judgments can be as large as \( n(n-1) \) when measuring the interdependencies between a quality attribute and its \( n \) contributors. If the number of contributors is large, the number of required comparisons will be quite large. This will inevitably lead to a scalability problem. In that case, the number of judgments required by one comparison matrix may be too large for a single domain expert in some industry-size case studies. Two or more domain experts are needed to make the pair-wise comparisons together for a single comparison matrix. The current quality modeling approach cannot check the consistency of multiple domain experts’ judgments.
• To measure the interdependencies between features and quality attributes, the proposed approach needs to find all the valid combinations of features which contribute to a quality attribute. However, the number of feature combinations in a feature model may be exponential to the number of features in a worst case scenario. Although the automated reasoning tools are used to accomplish this task, it may be still time-consuming for finding all valid feature combinations in industry-size case studies.
7. Conclusion

7.1 Summary

In the world of competitive business markets, software companies have to make sure that their users or clients can get competitive and effective software products faster. The software product line (SPL) approach emerged as an important software reuse paradigm which can help reduce time-to-market, decrease development cost and improve product quality. In an SPL approach, a feature model is used to represent the commonalities and variabilities among the product line members in terms of features. Although the feature models can be used in an SPL approach in different ways, the most popular use is to serve as a configuration model for deriving new products. The process of legally selecting the desired features from a feature model based on the customers’ requirements is called feature-based product configuration. Each valid combination of features derived in feature-based product configuration corresponds to a valid product.

Over the last decade, the researchers and practitioners in the software product line community have made great achievements on the techniques and tools for the automation, optimization and realization of feature-based product configuration. However, two practical problems are still not addressed properly and considered as the main obstacles of applying the feature-based product configuration in practice. Firstly, a feature model must be error-free for effective product configuration. The current validation approaches are very inefficient when dealing with large-scale feature models, as they transfer the validation problem into a constraint satisfaction problem (CSP) and use solvers to explore the entire space of the CSP. It may take an
unacceptable amount of time to validate large-scale feature models. Secondly, a feature model must contain quality information for the product quality assessment at the early stage of product development. The current quality attributes modeling approaches require real products which are difficult to obtain in practice or involve onerous domain experts’ judgments. From the above stated problems, four research questions are raised and these questions have been answered by this thesis as follows:

1. **How to develop a feature model validation approach that is more efficient than the current CSP-based approaches.**

   This thesis has proposed a more efficient validation approach based on the structural properties of feature models. It focuses on identifying and explaining feature model errors based on the contradictory feature relationships that cause the errors. It is found that, as the contradictory feature relationships are identified in feature relationship propagation, the solvers are not required by the proposed validation approach. The correctness and performance of the proposed approach have been evaluated by comparing the validation results returned from the built validation tool suite (FMVTool) and a well known CSP-based validation tool (FAMA). The experimental results show that the proposed validation approach is more efficient than the CSP-based approach on the given test cases.

2. **How to find a better way to explain the feature model errors.**
The proposed validation approach explains a specific feature model error by the contradictory feature relationships which cause the error. This validation approach further generates the minimal explanations of a feature model error by finding all the optimal solutions of removing all contradictory feature relationships which cause the error. It is argued that the minimal explanations generated by the proposed validation approach are more complete than the minimum explanations generated by the CSP-based approach. Furthermore, there is no overlap among the explanations generated by the proposed validation approach.

3. **How to better model quality attributes in a feature model.**

This thesis has developed an approach of modeling quality attributes in a feature model based on the domain experts’ judgments. The interdependencies between features and quality attributes are measured and a knowledge database called quality attribute knowledge base is generated to store the captured quality knowledge. As an analytic hierarchical process is used to assist domain experts to make judgments, the proposed quality modeling approach creates a reduction in domain experts’ judgments. The number of judgments is reduced from $2^n$ (in the worst case) to $n(n-1)/2$ and the time spent for a single judgment is reduced from minutes to seconds on average.

4. **How to enhance the feature-based product configuration process based on the captured quality knowledge.**
This thesis has proposed the quality-aware product configuration based on the generated quality attribute knowledge base. Three steps are required to configure the desired products from a feature model in a quality-aware manner: first, the customers’ requirements are validated to check whether there exist any conflicts in the customers’ quality requirements. Then, the features are selected or removed based on the customers’ validated requirements. Finally, the configured product is modified if it fails to satisfy the customers’ quality requirements.

7.2 Future Work

In addition to the research work discussed in this thesis, we have considered numerous improvements and evolutions that we were unable to realize within the time span of a Ph.D. We may or may not be able to realize these improvements in the future. However, we hope to inspire other researchers by listing these potential future research directions.

- Quality Oriented Software Product Line Architecture

This thesis concentrates on the research work on feature models. The software product line architecture (SPLA) is another important asset and it is the key to the success of an SPL approach. In the future, we aim to focus on quality oriented software product line architectures design to achieve the product-line quality attributes, such as variability, flexibility and reusability. The key issue of this task is to develop an SPL architecture description language to incorporate sufficient variability mechanisms to correspond to the variabilities in feature models. Our aim is to extend an existing architecture description language with a set of variability realization mechanisms, such as architecture reorganization, variant architecture component and
optional architecture component. Once the architecture description language has been obtained, the method of transforming feature models to SPLA models through a set of intermediate models will be proposed.

• **Quality Assessment in Later Stages**

The proposed quality modeling approach focuses on the quality assessment at the stage of feature-based product configuration. The quality attributes can be estimated or predicted for a configured product, but not for the final generated product. Furthermore, the proposed approach only deals with feature-sensitive quality attributes which will be affected by feature selections, such as security and availability. The quality attributes which depend on architecture designs or styles, such as memory size or CPU consumption, are not considered in the proposed approach. In this sense, the proposed approach can be considered as a starting point for quality assessment among the lifecycles of software development in a software product line and it should be followed up by more quality assessment activities in later stages, i.e., architecture derivation and component composition. Another colleague in my research group is working on the quality assessment at the architecture design level (Tan et al. 2012). We aim to find out how to relate the quality assessment at different stages and propose a more complete quality assessment approach which can cover all stages of product development.

• **Aspect-Oriented Techniques**

In the proposed quality modeling approach, we represent the interdependencies between features and quality attributes in an xml format (Section 4.3.7) and store the relationships
among different quality attributes in a quality attribute knowledge base (Section 4.3.8). It is believed that the above mechanisms are not the best for representing the complex relationships related with quality attributes, especially when transforming feature models to architecture or implementation levels. We are quite optimistic that some better mechanisms can be developed for representing the complex relationships. For example, aspect-oriented techniques provide systematic means for the identification, modularisation, representation and composition of crosscutting concerns. Early aspect focuses on managing the crosscutting properties at the early development stages of requirements engineering and architecture design (Araujo et al. 2005). As quality attributes are naturally crosscutting, we believe that the mechanisms of representing the complex relationships related with quality attributes can be improved by adapting aspect-oriented techniques. The idea of adapting the aspect-oriented techniques has been presented in my paper (Zhang 2010).

• Automated Reasoning on Feature Models

Since a new feature model is proposed to represent the quality attributes in this thesis, we believe that more work can be done for the automated reasoning based on this feature model. First, some new quality-related analysis operations can be carried out, such as whether a product with the desired quality levels is available in an SPL. The quality attributes knowledge base generated by the proposed approach can be used to support these operations. Second, the interdependencies between features and QA features can be transformed into some impact relationships. In that case, new propagation rules can be proposed to deal with the impact
relationships in the feature relationship propagation and the proposed validation approach can be extended to validate the feature models with quality information.

- **Practical Application**

  The approaches proposed in this thesis are evaluated by some empirical experiments. These approaches can be further refined and improved through the insights obtained from the practical application. By using the proposed approaches to solve the practical problems, the practical benefits of these approaches would be evaluated by the users.
Reference


Workshop on the Engineering of Computer Based Systems, Belfast (pp. 255-264).


