On the Quality of Feature Models

by

Christer Thörn

Department of Computer and Information Science
Linköpings universitet
SE-581 83 Linköping, Sweden

Linköping 2010
Abstract

Variability has become an important aspect of modern software-intensive products and systems. In order to reach new markets and utilize existing resources through reuse, it is necessary to have effective management of variants, configurations, and reusable functionality. The topic of this thesis is the construction of feature models that document and describe variability and commonality. The work aims to contribute to methods for creating feature models that are of high quality, suitable for their intended purpose, correct, and usable.

The thesis suggests an approach, complementing existing feature modeling methodologies, that contributes to arriving at a satisfactory modeling result. The approach is based on existing practices to raise quality from other research areas, and targets shortcomings in existing feature modeling methods. The requirements for such an approach were derived from an industrial survey and a case study in the automotive domain. The approach was refined and tested in a second case study in the mobile applications domain.

The main contributions of the thesis are a quality model for feature models, procedures for prioritizing and evaluating quality in feature models, and an initial set of empirically grounded development principles for reaching certain qualities in feature models.

The principal findings of the thesis are that feature models exhibit different qualities, depending on certain characteristics and properties of the model. Such properties can be identified, formalized, and influenced in order to guide development of feature models, and thereby promote certain quality factors of feature models.
This thesis is the conclusion of my five years of research education. There are several organizations and individuals that enabled the work described in this thesis and deserve thanks for it.

First and foremost School of Engineering, Jönköping University where I have been working at the department for computer science and electrical engineering. My supervision team consisted of professors Kurt Sandkuhl, Kristian Sandahl and Sture Hägglund. The supervision trio has over the years provided excellent insights about how to do research. Prof. Sandkuhl has generously allowed me to explore a quite wide range of research ideas and options, which I believe has made my research education very educational indeed.

My coworkers at the department have made it a very nice place to work at, but Thomas Gustafsson and Wolfram Webers (now at Jönköping International Business School) have also provided interesting input and meaningful contributions to the research, for which I am very grateful.

Part of the work was conducted in the research project Semantic Structuring of Components for Model-based Software Engineering (SEMCO) supported by a grant from the Swedish Knowledge Foundation (grant no. 2003/0241). Other parts were conducted at Tactel AB, and I want to thank Marcus Stjärnås, Linus Stenbacka, Patrik Rostedt, Pernilla Selskog and all the workshop participants for their efforts and interest.

Christer Thörn
Jönköping, 2010
## Contents

1 Introduction  
1.1 Background and motivation ................................... 2  
1.2 Problem formulation .......................................... 3  
1.3 Research questions ........................................... 6  
1.4 Contributions .................................................. 6  
  1.4.1 Academic contributions ................................... 6  
  1.4.2 Engineering contributions ................................. 7  
1.5 Scope and intended readers .................................... 7  
1.6 Summary of publications ...................................... 8  
  1.6.1 Publications related to this work ......................... 8  
  1.6.2 Other publications ..................................... 9  
1.7 Thesis outline ................................................ 10  

2 Foundations of Feature Modeling and Quality  
2.1 Software modeling ............................................ 11  
2.2 Feature modeling ............................................. 15  
  2.2.1 Feature modeling practices ............................... 15  
  2.2.2 Feature modeling research ............................... 35  
2.3 Software quality ............................................... 36  
2.4 Quality of software models .................................. 39  
  2.4.1 Measuring model quality ................................ 40  
  2.4.2 Evaluating model quality ............................... 42  
2.5 Chapter summary ............................................. 43
3 Research Approach and Process

3.1 A view on software engineering research

3.2 Issues in model quality research

3.3 Survey methodology

3.4 Case study methodology

3.5 Experimentation in software engineering research

3.5.1 An experiment in feature modeling

3.6 Validation in software engineering research

3.7 Research process

3.8 Chapter summary

4 The Need for High Quality Feature Models

4.1 Current practice in feature model development

4.1.1 A selection of feature modeling methodologies

4.1.2 Criterion 1: Reuser involvement

4.1.3 Criterion 2: Scenario consideration

4.1.4 Criterion 3: Feature model validation and verification

4.1.5 Criterion 4: Quality attributes considered

4.1.6 Criterion 5: Meta-models and notations

4.1.7 Summary

4.2 Survey on industrial state and needs

4.2.1 Survey participants

4.2.2 Instrumentation

4.2.3 Validity and appropriateness

4.2.4 Respondents’ characteristics

4.2.5 Product variability

4.2.6 Problem space variability

4.2.7 Solution space variability

4.2.8 Perceived needs for changes and improvements

4.2.9 Contrasts with previous findings

4.2.10 Summary

4.3 SEMCO

4.3.1 Context

4.3.2 Objective

4.3.3 Brief results and value

4.4 Satisfying the need

4.5 Chapter summary
5 Initial Quality Model and its Application 105
  5.1 First iteration of the quality model .......................... 106
    5.1.1 Overview ............................................. 107
    5.1.2 Qualities and quality attributes ......................... 110
    5.1.3 Indicators ............................................ 115
    5.1.4 Relationships and trade-offs .......................... 116
  5.2 Case study: Tactel AB ...................................... 116
    5.2.1 Paired comparisons ................................... 116
    5.2.2 Case study context .................................... 117
    5.2.3 The company .......................................... 118
    5.2.4 Objective ............................................. 119
    5.2.5 Planning and preparation ............................... 120
    5.2.6 Scoping ............................................... 121
    5.2.7 Tooling (FMWIKI) .................................... 122
    5.2.8 Prioritization ........................................ 124
    5.2.9 Construction ......................................... 125
    5.2.10 Evaluation .......................................... 126
    5.2.11 Comparison of prioritization and evaluation ........... 127
    5.2.12 Indicators and interpretations ......................... 128
    5.2.13 Discussion of case study interpretations ............... 137
  5.3 Example values of the indicators ............................. 143
  5.4 Chapter summary ........................................... 144

6 Refined Quality Model and Approach 145
  6.1 Second iteration of the quality model ....................... 145
  6.2 Prioritizing the quality of feature models ................. 150
    6.2.1 Scenario ............................................. 150
    6.2.2 Procedure ........................................... 151
    6.2.3 Instrumentation ....................................... 152
  6.3 Evaluating the quality of feature models .................... 155
  6.4 Comparison ............................................... 155
  6.5 Constraints ............................................... 157
  6.6 Development principles .................................... 157
  6.7 Integration and operation ................................... 161
    6.7.1 The quality model and the feature modeling process .... 161
    6.7.2 The quality model in the development process .......... 163
  6.8 Chapter summary ........................................... 164
7 Discussion  165
  7.1 Research results and improvements  . . . . . . . . . . . . . . . . . . . . . . . . . 165
    7.1.1 Have the research questions been answered?  . . . . . . . . . . . . . . . . . . 167
    7.1.2 Could we have learned the same with less effort?  . . . . . . . . . . . . . . . . 168
    7.1.3 What could we have learned in addition?  . . . . . . . . . . . . . . . . . . . . 169
    7.1.4 Are the results scientifically grounded?  . . . . . . . . . . . . . . . . . . . . 169
    7.1.5 Did we pass the test of reality?  . . . . . . . . . . . . . . . . . . . . . . . . . . 170
  7.2 Requirements revisited  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 170
  7.3 Future work  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 171

8 Conclusion  175
Chapter 1

Introduction

Judging by the amount of books, conferences, publications, and experience reports available, one of the foremost means of modeling the structure of families of software products is to use feature models [7, 14, 22, 25, 32, 38, 39, 46, 51, 121, 10, 70, 62, 72, 90, 98, 102, 103, 108]. Feature models describe the common and variable parts of a software family in terms of features. Each feature in the model represents a meaningful function of the software family, often with the intent to have features on a higher level of abstraction than requirements. The value added by a feature should preferably be usable by a wider range of stakeholders compared to requirements, so a feature can aggregate several requirements, or cross-cut different requirements. The features in a feature model are connected in a graph to show the dependencies and relations between them. By combining features according to the constraints set by the structure and relations in the model, products that provide functionality targeted to a particular market or specification can be composed and configured.

Feature modeling gained momentum in the 1990s and has since been the subject of significant amounts of research [13, 122, 32, 38, 49, 51, 57, 70, 99, 73, 123]. There are now many methods described for developing feature models and there are many notations and meta-models to choose from when developing and visualizing feature models. However, there is little research done on the topic of the quality of feature model, i.e. how well the produced feature models suit their intended purposes. There are many ways described to produce fea-
ture models, but very little on how to ensure, or know, that the models that are produced also are usable, correct, and suitable.

The work described in this thesis focuses on concepts and methods for quality management – including quality model definition, quality design, and quality improvement – of feature models.

1.1 Background and motivation

Today a wide variety of products and services are relying on software as a critical component. Software adds functionality, interfaces, possibilities for personalization, etc. In many cases, software is used to provide variants of the products, suiting different requirements and market needs. When the hardware parts of the products often provide fixed limits to variability, software takes the role of variability conduit [4, 8, 11, 17, 24, 122, 33, 45, 56, 63, 6, 69, 75, 78].

Customers of software-intensive products and services expect to get products that are highly suitable for their particular goals and contexts. The competitiveness in the software industry means that modern software often must be usable in many different versions and variants of the software, each variant providing a particular set of functionality. Although the need for flexible and highly capable software keeps rising, the age-old desires of high quality, cost efficiency and ease of maintenance still remain.

Many organizations find themselves in a situation where the need for higher degrees of variability in the software is becoming a problem because of the configuration issues, risks of duplicated work, problems with communication, etc. [117]. The variations and concurrent versions of the software designed for different end-user products tend to become so prolific that overview and control are lost, with duplicated work efforts, quality loss and other problems as a result. With growing families of products, different series, individual variations and customizations, a proper management of variability of software becomes vital to organizations with expanding product portfolios.

Moreover, modern, complex systems with high variability can pose a difficult challenge to their developers. It is desirable to provide high levels of flexibility in order to have variants that satisfy the different needs and application contexts that exist. At the same time, the systems must be sufficiently restricted and controlled in order to be maintainable and manageable. An illustrative example of the challenge comes from the automotive industry. In the past 20 years, the electric/electronic sub-systems in a normal car has more than tripled in numbers\(^1\).

1.2 Problem formulation

Many of the sub-systems are available in different versions to satisfy different market needs and customer wishes. The number of combinations is immense.

Modeling a system or entity can give us an increased understanding of problems, and let us test designs and concepts used for different solutions, before devoting resources to implement an actual artifact. Variability modeling is about analyzing the structure of common and variable parts for product families, product lines or other systems that come in several versions and share common parts.

Among different notations and methodologies for modeling variability, feature models are, arguably, one of the more common. Figure 1.1 shows a simplified example of the principles in feature models. The root feature denotes the main concept of the feature model. Subsumed under the root feature are mandatory features that are included with all variants in the product line, optional features that can be included if wanted, and alternative features provide feature sets to choose suitable options from. Using constraints, various relations and compositions can be supported.

By modeling the structure and dependencies of software parts and components used in several different products, several benefits can be achieved. Among them, the construction of a software product line architecture to leverage reuse of existing assets, an understanding of dependencies and influences between communicating interfaces, a comprehensive and communicative overview of product structures, and a support for tracing change requests and modifications throughout reusable modules and assets in the software system.

In the end there are benefits in reuse of assets, higher utilization of invested efforts, reduced risk of duplicating work, and possibility to better coordinate product offerings to be gained from modeling the variability structure. But only if the feature models produced are well-made and suitable for their intended purpose. The goal for the work in this thesis is thus to look into means for ensuring and improving the quality of feature models, where quality is seen as suitability and fitness for a designated purpose.

1.2 Problem formulation

Small and medium-sized enterprises (SMEs) constitute a substantial segment of the software developing industry [37]. More and more, they find themselves facing variability management problems for their products [3, 86, 117, 123]. While feature modeling is an interesting opportunity for many – because of its ability to bring structure to complex product families – there is also a cost and an investment of effort associated with the construction of feature models. In order to make sure that the effort is well spent, the organization should be able to direct the efforts and resources in a way that the modeling outcomes are likely to be well
sured for the purposes, goals, motivations and contexts that the organization is working in. Modeling generally uses experts and experts’ knowledge to elicit the information that is needed for the model. The availability of experts’ time is often limited, so being able to use that time as efficiently as possible is very important.

Apart from the different goals and motivations that users of variability modeling have, there are also considerations for different domains to take into respect. For example, organizations working in safety-critical applications, embedded applications and end-user configurable applications have some goals in common, but also differing needs and requirements on the modeling results. These settings and contexts pose further demands on what is needed from variability models.

Figure 1.1: An example of a partial feature model describing a car.
The key is to design the models with quality in mind. To explore and understand the relations and connections between the focus of the efforts put into modeling, and the results of those efforts, is a key concern in order to provide means for effectively directing and guiding the modeling efforts, so that the result is perceived as appropriate and useful. Hopefully, an understanding of feature model quality increases the chances of achieving the purpose of a specific model. Given a context and a usage scenario of a feature model, goal formulation and prioritizations can be made in order to ensure that the elements and properties of the feature model that are most important for the context and usage are given appropriate attention.

The work reported in this thesis includes technical aspects such as quality factors and dependencies between qualities, organizational aspects such as application and evaluation of feature model quality, and empirical aspects such as determining how quality factors are influenced by properties of a feature model.

The technical aspects involve the selection and modeling of properties and traits of feature models, and understanding how they are related to each other. The underlying assumption is that there are some aspects that influence the quality positively and other aspects that have a negative impact on the models’ quality. Researchers have spent quite some effort on trying to determine how properties of software models influence the usability of the models. Many different kinds of metrics and measures of software models have been proposed, including structural and layout-oriented metrics [67, 43, 82]. The origin of such metrics and measures are the models of quality attributes used to determine software quality. This thesis shows how such quality attributes can be applied to feature models and how the dependencies and influences are affecting the quality attributes.

The organizational aspects concern the processes and activities necessary to allow an organization to draw benefits from feature modeling. Not only the activities carried out to perform the modeling in itself affect the resulting modeling artifacts, but also the structures and activities within the organization that are relevant for the variability of the products, problems and solutions. The deployment and application of the model are based on the organizational goals for the variability modeling, and the settings and contexts in the organization influence the modeling procedure and outcome. Learning and knowledge transfer aspects are also important to treat in a systematic manner, through evaluation and identification of strengths and weaknesses in the procedures and results of the variability modeling.

The empirical aspects of this work analyze the indicators for each quality on practitioner-oriented cases and samples. The properties of a model that leaves a positive or negative contribution to the emergence of a quality factor is here
1. Introduction

termed an indicator. If a number of indicators can be decided for a quality factor, the indicators can be changed and modified in order to change the prominence of the particular quality. This would allow the qualities that are regarded as the most important for a certain feature model to be emphasized at the expense of less prioritized qualities. Finding the indicators and their influences need to be done by gathering empirical material, where practitioners and applied feature models are sources for the analysis.

1.3 Research questions

The specific research questions that we wish to provide answers for in this thesis can be formulated thusly:

- Do properties of a feature model affect its usefulness in different contexts?
- How can a feature model’s properties be evaluated with regard to their effect on the quality of the feature model?
- What properties of a feature model can be changed or modified to influence the quality of the feature model?

The research questions are formulated under some assumptions. For one thing, it is assumed that it is possible to define, to sufficient degrees, what the purpose of a feature model is. It is also assumed that there are in fact meaningful differences between models that make users of the models prefer one or the other for a particular usage scenario. The term feature model is used in a wide sense in this thesis. More on our feature model definition is found in subsequent chapters.

1.4 Contributions

Like most research results in software engineering, the contributions of this thesis are of both academic and more practical nature. We present the main contributions for a software researcher’s and a practitioners perspective.

1.4.1 Academic contributions

- A model of qualities, indicators and dependencies for feature models.
- An approach for finding the desirable and non-desirable qualities and properties of a feature model.
- An approach for evaluating the quality of existing feature models.
1.5 Scope and intended readers

- An investigation into the industrial needs for variability management and modeling.
- An evaluation of how existing approaches for development of feature models take the quality and properties of the feature model into regard.

1.4.2 Engineering contributions

- An instrumentation and practice for finding the desired and actual qualities for a feature model.
- An initial set of development principles for designing feature models that exhibit particular qualities.
- Empirically grounded considerations to equip an existing method for feature model development with.
- FMWIKI, a web-based feature modeling tool used during a second case study.

1.5 Scope and intended readers

There are two terms in the title of this thesis, that can be the subject of lengthy debates on their own; feature models and quality.

The plethora of different definitions and versions of feature models and the feature concept warrants a comment on the scope of this thesis. There is much common ground between the different kinds of feature models, and it has been these common traits that have been used for the understanding of feature models during this work. In the next chapter, several – more or less – popular methods and notations for feature models are described, along with the interpretations of the feature concept that have proven most useful in the experience of the thesis author. While the designers and inventors of the different feature model versions will certainly be able to point out many differences and deficiencies in the other versions, the results in this thesis have not been intended to depend on a particular feature modeling approach.

As for the term quality, a constraint on the work reported in this thesis is that it cannot claim to be a complete and final discussion on the quality of feature models. We do not know how many different influencing properties there are to investigate, how many of those properties that can be modified to reach different resulting qualities, etc. In this thesis we include a set of six principal quality factors, and a set of indicators for those qualities that are found through empirical studies. The proposed set of quality factors are sufficient to cover the aspects and
influences that are most commonly regarded by other published quality models for related subjects.

The contributions of the thesis are intended for several groups of readers. The academic contributions are mainly aimed at researchers in feature modeling methods and methodologies, and researchers in model quality. The results from the industrial needs on variability management practices can be of interest to researchers that are curious of the practices in small and medium-sized enterprises (SMEs) since they were the focus of the investigation. With the results reported in this thesis in mind, further development of feature modeling practices and methodologies can hopefully take into consideration both the aspects of quality of feature models, as well as the requirements of SMEs.

The engineering contributions are oriented to practitioners of variability modeling. Organizations that consider variability modeling as a means to get better overview and structure of their product portfolios can benefit from reviewing the procedures and guidelines described in later chapters. Practitioners should hopefully be able to better direct their modeling efforts in order to achieve their goals with the model and relate the experiences described in the case studies to their own context.

1.6 Summary of publications

Parts of the work related to this thesis have been reported in a number of publications. Some parts of the work described in the thesis have not yet been published elsewhere, or were part of non-public project reports. Additional papers have been written or co-authored by the author, but are less important to the main theme of the thesis.

1.6.1 Publications related to this work


  The contents of this publication is complementary to the descriptions of the industrial case study found in the second half of chapter 5.

1.6. Summary of publications

The contents of this publication describe the industrial needs discovered through a survey that is detailed in the middle of chapter 4.


The contents of this publication is essentially the same as the section about feature modeling found in the first half of chapter 2.


This publication describe the initial model for feature model quality that is seen in the first half of chapter 5.


This publication investigates the possibility of evaluating the quality of variability modeling languages. It has not been included in this thesis.

1.6.2 Other publications


### 1.7 Thesis outline

Chapter 2 presents the preliminaries of software modeling, feature modeling and software quality that are referred to throughout the rest of the thesis. It contains a primer to feature modeling notations and methods, and also cover other work related to evaluating the quality of models.

Chapter 3 covers the research methodology that has been used during the thesis work. It provides an overview of important issues related to research on model quality, and how they have been addressed in this work. It also describes different approaches to gather empirical material, and some experiences from using an experimental approach.

Chapter 4 provides the motivation for the work conducted for this thesis. It describes an evaluation of current feature modeling practices and how they consider the quality of produced feature models. It then accounts for the state of practice and the needs perceived in industry, by means of a survey of SMEs. An initial case study and its contributions to the motivation is described. Finally, a list of requirements on an approach to support higher quality feature models is presented.

Chapter 5 describes the first iteration of a quality model for feature models and the quality model’s different elements and its application is detailed. The initial model was based on experiences in the initial case study, and evaluated in another case study, presented in this chapter.

Chapter 6 presents a refined, second iteration of the quality model and its elements. Updated or added procedures for prioritizing, evaluating, developing and deploying feature models of increased quality are described.

Chapter 7 contains a discussion on the research results and the improvements that could have been made in retrospect. The research questions are revisited and possibilities for future work are described.

Chapter 8 concludes the thesis, summarizing the contributions of the thesis and their intellectual and practical merits.
This chapter provides an introduction to the concepts and theories underlying the work presented in this thesis. An assumption of the work, based on experience reports over the last decades, is that modeling gives meaningful contributions to the software development process. We introduce fundamentals of modeling in general, software models, and the special case of feature models. The premise of the work is to increase the quality of feature models, so we also cover the basics of software quality and the use of quality models. The application of quality models and approaches to software models is covered to provide perspective on quality of feature models.

2.1 Software modeling

A model is, in its most abstract sense, described as a conceptual representation of some process or interactions, or a theoretical construct of a phenomenon. In the engineering principles and in the natural sciences, we usually find models as...
being abstracted, simplified and idealized constructions of, possibly non-existing, real-world concepts or processes. In [74] it is pointed out that a strength of models is the power of abstraction, for instance letting a toy-car model not represent just one single car, but all cars of that particular make, essentially modeling a class of objects.

Models in engineering come in many domain specific forms. Construction engineering use building blueprints, mechanical engineering use plans, and electronics engineers use circuit schematics to describe the solutions within their line of work. In all cases the purpose of using the models is similar. They assist in the development, manufacturing, and maintenance of products or artifacts by providing information about the consequences of building the artifacts before they are realized [74]. Modeling techniques also support the planning and structuring of the development activities [41].

The benefits of using models in development and engineering work are, for example, the possibility of having a formal and common understanding of concepts, processes and behaviors concerning a particular subset of our world. This understanding can be reached between all the stakeholders of the modeled concepts. Models can act as a semi-formal language, and help elicit requirements. Models allow us to represent problems and solutions to problems in an implementation-independent way, using abstraction and simplification. They allow the developers to simulate and evaluate how design decisions will affect the artifact, without having to take the risks that might be associated with a full-scale experiment. They also collect and provide information that can be stored and retrieved later at a point where it might be unpractical to acquire the information through interactions with the real artifact. Obviously, models as most people know them also provide an alternative way of presenting essentials of artifacts, most notably through visualization. A human tends to have vastly greater understanding for a visual representation of a modeled subject, compared to for instance tabular data.

According to [109] a model should meet three criteria:

- Mapping, meaning that there is some original object or phenomenon that the model is a representation of.

- Reduction, meaning that the model does not reflect all properties and attributes of the original. However, the model has to mirror some of the attributes of the original.

- Pragmatic, meaning that the model serves as a meaningful and usable replacement for the original in some context, or for some purpose.
2.1. Software modeling

The mapping criterion does not imply existence of the original. For instance, a cost estimation or project plan is speculative concerning the actual costs or time planning. The reduction of the original to the model means that many attributes of the original are waived in the model. On the other hand, other attributes are added to the modeled object as abundant attributes that are not reflected in the real object (Figure 2.1) [109, 74]. In this vein we could consider the descriptions true of the original or world as DW, and the descriptions true of the software or the product as DM, in which case the modeling relationship are those that are in their intersection (Figure 2.2) [55].

Models come in two principal flavors, descriptive models which mirror an existing original, or prescriptive models which can act as an instruction for constructing an original according to the model. Descriptive models do not always have to be created after the original has been created, since a prognostic model, such as a cost estimation, are descriptive as they cannot (directly) influence the original. [109, 74]

In software engineering, most models are prescriptive. [74] lists process models, information flow models, design models, user interaction models, models of principles such as patterns and process maturity models. Obviously each of these
cases can be refined and further specified into the various models and constructs that exist within each of the many modeling paradigms and notations that exist.

The position of models in software engineering is quite strong, and rightfully so. In a sense, software itself is very often a model of the world. Unlike traditional engineering disciplines, the artifacts that are resulting from software engineering are themselves models of complex processes, relations and objects. Due to the inherit complexity of the real world—which software models—software engineers use models to achieve reduction of complexity and abstraction of concepts, both of which are necessary to reach sufficient succinctness to be able to communicate the ideas set forth in the model to the stakeholders of a software development project. However, there is a common misconception that software models are primarily documentation artifacts and the development and use is only peripheral to software development [42]. This has resulted in many discussions on the value of software modeling having instead concerned the value of documentation in software development.

Over time several modeling paradigms have come up. A significant and influential advance in software modeling is object-orientation, which has by now become more or less ubiquitous and is considered the de facto way of modeling and implementing software intensive enterprise systems. It is worth noting that the world is in fact not object-oriented. Although many of the modeling paradigms are powerful and applicable in many situations for modeling problems and solutions, models are unlikely to be adequate for a correct formalization and will only serve as a complement.

The uptake of software modeling practices in industry ranges from entirely model-based development approaches to none. Many organizations see benefits of raising the abstraction level of the software from code to models, and find adequate tool support to do so. However, there are also organizations that perceive obstacles to using models in their development processes. In the survey carried out by the author [118], seen in section 4.2, the SMEs that responded found modeling practices to add an overhead that displaced the benefits. Changes made on a regular basis were too small to warrant a model-based approach. Direct applicability of the models is seen as important, the most common models being use-cases and flow charts. Nevertheless, modeling is seen as quality improving as they provide a higher level of formality and result in fewer errors due to miscommunications.

According to France and Rumpe [42], current research in software modeling tends to focus on generating implementation and deployment artifacts from detailed design models of software. The model-driven development technologies are mostly concerned with using models for generating code in regular pro-
Feature modeling offers an interesting contribution to solving the problem of capturing and visualizing commonalities and dependencies between features and between the components providing feature implementations. Since the 1990s, feature models are frequently used to model software product lines and software families, often in concert with software architecture practices. This section introduces feature modeling practices and fundamentals, and ways of integrating feature modeling into software development processes. These topics were treated in [120].

2.2.1 Feature modeling practices

The purpose of a feature model is to extract, structure and visualize the commonality and variability of a domain or set of products. Commonality are the properties of products that are shared among all the products in a set, placing the products in the same category or family. Variability are the elements of the products that differentiate and show the configuration options, choices and variation points that are possible between variants of the product, intended to satisfy different customer needs and requirements. The variability and commonality is
modeled as features and organized into a hierarchy of features and sub-features, sometimes called feature tree, in the feature model. The hierarchy and other properties of the feature model are visualized in a feature diagram, see for instance Figure 1.1 for a simple example. More examples of syntax for feature diagrams is explained further ahead in this chapter.

Domain Engineering

The roots of product line approaches, software families, and structured reuse of software assets in different variants, is found in the ideas of domain engineering and application engineering.

While conventional systems engineering aims at satisfying the requirements for single systems, domain engineering produces assets usable to satisfy the requirements of a family of systems. [22] refers to areas organized around particular classes of systems as vertical domains. Parts of systems organized by functionality are referred to as horizontal domains. For software systems, domain engineering along a horizontal domain would result in reusable software components for user interfaces, communications, etc. that are shared across several products in the software family. Domain engineering along a vertical domain would result in frameworks that could be instantiated to produce any system in the domain. The modeling of vertical domains should be done using several horizontal domains.

Figure 2.3 illustrates the connection between system engineering and domain engineering. System engineering consists of several phases as described below and illustrated in boxes 7 to 12. The artifacts developed during system engineering are subject of reverse engineering activities in order to derive contributions to the domain model. Domain engineering aims at developing and maintaining artifacts that can be basis for system engineering projects. This is illustrated in boxes 1 to 6 and described in the following section. Supporting functions for both system and domain engineering are shown in the middle of the figure, and include change management and configuration management. The two workflows are concurrent and while it would be preferred if one started with domain engineering before application engineering, it is often the case that existing products and assets need to be reverse engineered to domain assets.

The Domain Engineering workflow

The domain engineering workflow consists of three major phases, domain analysis, domain design and domain implementation, each described below. The domain analysis is the fundamental activity in domain-based software reuse and is typically the activity that initiates the whole process of adapting an organiza-
2.2. Feature modeling

The first part of domain analysis is domain scoping. During this activity, the focus, boundaries and limits of the domain to be modeled are appropriately defined. While not making the scope too wide, which would reduce the chances of the organization being viable and able to successfully conduct their business, the scope has to accommodate the potential of the domain in the future. It is important to use a scope that allows for sufficient flexibility in the products that are to result from the development, but not let the scope of the domain stray so that the core assets cannot accommodate the products. This would lead to a return to the classical development of one product at a time and one would loose the benefits that one hopes to achieve through software reuse. The scope should also identify the stakeholders and their interests that influence the domain. The stakeholders of the domain include managers, developers, investors, customers, end-users and so forth. It is argued that the delimitation of the domain is in fact the range of interests that the stakeholders have. The scope evolves as changes in market, organization, and environment comes about. This could typically be part of continuous business analysis activities.

Figure 2.3: Overview to Systems and Domain Engineering. [36]
Scope is determined on the grounds of marketing reasons and on technical basis. While a set of products might together satisfy a particular market segment, there could as well be a sensible set of products sharing technical characteristics that would make for a good software family. The term product family traditionally refers to a group of products that can be built from a common set of assets, based on technical similarity, whereas the term product line refers to a group of products sharing a common set of features that satisfy a need, based on marketing strategy. The distinction between product lines and product families is quite blurry and the terms are sometimes used interchangeably in literature.

The second part of domain analysis is the domain modeling, in which the domain model(s) are produced. The domain model is the explicit representation of the properties, semantics and dependencies of the concepts in the domain. It would typically be a set of different components and models, each describing one or more aspects of the system from a domain modeling perspective. Rather than all domain knowledge to be contained in one single model using a particular modeling language and notation, the strengths of a variety of modeling languages models can be utilized.

The following components of a domain model are listed by [22]:

**Domain definition**  Defines the scope of the domain in order to determine which systems that are encompassed by the domain and the rationale for inclusion and exclusion of systems for the domain.

**Domain lexicon**  A taxonomy defining the domain vocabulary as it is understood by the practitioners in the domain.

**Concept models**  Various models used to describe the concepts in the domain formally. This means models such as class diagrams, interaction diagrams, etc. Apart from formal models, this could also include informal textual descriptions.

**Feature models**  [22] puts emphasis on feature models as an important contribution to domain modeling and places feature models outside the other concept models. Feature models describe the meaningful combinations of features and functions of the products in the domain, thus the commonality and variability of the software family.

Domain design is the subsequent activity of domain analysis that takes the domain model, and develops an architecture and production plan for the family of systems to be built from the assets in the domain. The architectural design resulting from this activity prescribes how the components and assets are to be assembled to satisfy the requirements that can be posed on the family.
architecture has to be constructed in order to accommodate all the variability possible in the family. Since the architecture is a description of the components available in the system family and the composition constraints placed on their interactions, one can see a close connection to the descriptions in the feature model. The architecture should not only consider functional descriptions, but also non-functional requirements, such as performance, compatibility and so on.

The second artifact of the domain design activity is the production plan which describes the process of how to assemble the components, how to handle change requests, custom development and adoption of the assets to special requirements and the evolution of the measurements and processes used in the development of incarnations of the family.

Once the domain design has been completed, it is followed by the domain implementation phase which involves implementing the components, languages, development processes and other assets designed. It also involves building the infrastructure used to realize the reuse of the assets in the domain model. That is, the interfaces and storages to find, access and assemble the components to instantiations of the product family.

**The System Engineering workflow**

The system engineering workflow builds products and configurations of the software family using the reusable assets that results from the domain engineering phases. The system engineering workflow is intended to be carried out in parallel with the domain engineering activities, but while there is one instance of the domain engineering workflow for each product family, the system engineering workflow exists in several instances, one for each product to be produced in the software family. Figure 2.3 should thus not be interpreted as if the system engineering and domain engineering activities are running synchronously with each other, but rather illustrates the flow of assets, results and information between the phases involved.

The domain engineering workflow is iterative and constantly updates the assets and expands the capabilities of the organization with regard to what sort of products the organization can provide. The process of developing systems based on the reusable assets is iterative as well, not only in that the product is released in new and updated versions i.e. maintenance, but also in the application of iterative software development processes to the development of the systems. For instance, the system might be developed using prototyping, where new versions of the prototype system are produced as assets are implemented in the domain implementation phase.
The phases in system engineering correspond well to the ones that we find in most single system and software development methodologies [92]. The initial requirements analysis takes the customer requirements and matches them to the set of capabilities that the software family can fulfill using the domain model. The requirements that cannot be fulfilled using the resources in the domain model and domain assets are fed to the domain analysis phase in order to determine whether those requirements should be accommodated in the software family. If that is the case, the domain engineering activities create the reusable assets to satisfy the requirements and store them to be used for future system engineering activities. While the domain engineering approach to reuse of software intends to use the assets to the greatest extent possible, the software family assets cannot possibly accommodate every configurability option. There will inevitably be concrete customer requirements that cannot be fulfilled using the resources from the family domain engineering, and which will not be suitable for inclusion in the domain model or the reusable assets, being specific to the particular product. These requirements will require and trigger custom design and development specific for the current product.

The result of the system design phase is the software architecture that will be used to accommodate the reusable assets that fulfill the customer and system requirements. At the last leg of system engineering, the actual product is configured and instantiated using the reusable assets. The assembly of the assets can be manual, automated, or use a semi-automated approach. Depending on how suitable the software components are and how developed the organization is, the instantiation of the assets could be done using generators, code configurators or other advanced techniques.

A Brief History of Feature Modeling

The subject of feature modeling is tightly connected to domain analysis. The intention of analysing and modeling a domain is that as an organization constructs systems and conducts its business, it gathers experience and know-how. As most systems constructed in the organization are likely to share technical characteristics and are designed to meet similar requirements, it is likely that the organization can benefit from the acquired knowledge, as subsequent systems are constructed.

The idea of domain analysis was introduced in the work conducted on software families in the mid 1970s. The term domain analysis was coined in 1980 by Neighbors [85] and described as “the activity of identifying the objects and operations of a class of similar systems in a particular problem domain.” Neighbors expresses the key idea of domain oriented reuse in that “it seems that the key to reusable software is
2.2. Feature modeling

to reuse analysis and design; not code.” The original idea of what to be captured in the domain model resulting from the domain analysis was refined and revised with the methodologies developed. Where the basic idea of “objects and operations” from Neighbors remain, the advent of more advanced modeling tools, object-oriented modeling and other modeling methodologies meant that the domain model could be equipped with more advanced constructs, such as use-cases, feature models and concept models like class diagrams etc. [19]

A lot of work in the field of domain engineering and domain-based reuse of software was conducted in the research programs sponsored by the US Department of Defense, called Software Technology for Adaptable Reliable Systems (STARS) [18] and sub-projects such as Central Archive for Reusable Defense Software (CARDS) [5]. These research programs spawned several variations and directions, among those the Software Engineering Institute’s (SEI) Software Product Lines (SPL) [17], ODM [105] and several other methodologies and guidelines for domain analysis.

The concept of feature models was originally introduced in 1990 by Kang et al. with the Feature-Oriented Domain Analysis (FODA) technical report [66]. The original use for feature models was to facilitate domain analysis in telecommunications systems, but has since been used in many other domains. The original definitions, notations and concepts used by FODA have been extended and modified as various other uses for feature models have become apparent. Many examples of requirement abstraction, architecture specification, etc. have been put forward over the years [51, 72].

FODA was followed by the successor FORM (Feature-Oriented Reuse Method) [62]. A significant leap towards formalizing feature models was taken by Czarnecki and Eisenecker in Generative Programming [22], and further refinement of the feature model notation was published by Riebisch et al [99, 98].

Using features as a way to describe software and system functionality is considerably older than FODA, although it was FODA that introduced structured modeling of features. The idea was incorporated in many of the techniques for domain modeling and domain analysis which appeared during the 1990s, some of which briefly flashed into existence and went away, while others became successful and are used to this day. Today, feature modeling is an established and widely used technique which has been incorporated in many development methodologies and is the subject for interesting research.

The original use for feature models as a means and aid to perform domain analysis was over time complemented with other uses, since the structuring of properties of a domain or product set into features, turned out to be an efficient and communicative representation. Apart from domain analysis, there are two principal categories of uses, which most efforts can be sorted under.
2. Foundations of Feature Modeling and Quality

The first is to support the requirements management process for lines or families of products. By letting the capabilities of a product line be represented by features, requirements posed on the product by customers, as well as internal requirements, can be abstracted into features. By letting a set of common requirements on a product be represented by a feature, one can achieve higher degrees of reuse. Using this approach, feature models can be used as a means of communicating information about the product to customers during requirements negotiation or sales, conveying implementation requirements to developers, or a means to exchange information between other stakeholders.

The second main use for feature models is configuration and automated construction of an instantiation from the product line described by the model. Ideally, this would mean that each feature would principally represent one or more components or source-level packages used to add functionality to the instance. The selection of features from the feature model would thus guide automated scripts, which would build the product with the requested functionality.

The Feature Concept

The main concept of feature models is of course features. There are several definitions of the term feature used in conjunction with feature models. Some of them are more formal, while others are more intuitive. Features are intended to be concepts described by a single word or short line of text. These are some definitions found in literature:

- From FODA by Kang et al: “A prominent or distinctive and user-visible aspect, quality, or characteristic of a software system or systems.” [66]
- From Czarnecki and Eisenecker: “Property of a domain concept, which is relevant to some domain stakeholder and is used to discriminate between concept instances.” [22]
- From IEEE: “A software characteristic specified or implied by requirements documentation (for example, functionality, performance, attributes, or design constraints).” [53]
- From Riebisch et al: “A feature represents an aspect valuable to the customer.” [98]
- From Bosch: “A logical unit of behavior that is specified by a set of functional and quality requirements.” [11]

The characteristic of user-visibility is interesting, as it places constraints on what should be considered features. One can argue that both the definitions that considers user-visibility and the ones that do not are equally valid, if viewed from different perspectives. From an applied, practical and user-oriented view
it is meaningless to include features in a model that would not add to the users perception of the product [15, 70]. The view of domain engineering would on the other hand want to include as much as possible of the domain information in a model, and would thus like to include all information relevant to any stakeholder. When discussing products that are part of larger systems in general, and perhaps embedded systems in particular, the latter perspective serves better.

While the original semantics of feature models was not very well-defined to begin with, further extensions and modifications of the feature models have resulted in more and more semantic meanings being imposed on the relations, without really clarifying much. As the original tree structure is pushed back, in favor of more general directed graphs with equivalent types of edges, the matter of clearing up the semantics is an interesting research question, albeit on the theoretical level.

In Section 6.1, there is a meta-model for feature models described that was used during the application of the work in this thesis. There has however not been a particular feature modeling approach or definition used as a prerequisite during the investigations of this thesis. Out of the definitions mentioned above, there is nothing that is directly contradicting, and all the definitions fit the understanding of the feature concept that has been used during this thesis. As long as a feature express a trait that provides variable functionality or properties to the product, and is on a different abstraction level than a requirement, it is compatible with the both the popular views as well as this thesis’ view.

Feature Diagrams

The feature diagram is an essential part of a feature model. The feature diagram is the visualization of the feature model. It is a hierarchical decomposition of the features in the model, indicating dependencies and constraints for the commonality and variability of the product that is represented by the feature model. It usually has the form of a (connected) tree structure with the root node of the tree representing the concept that is described by the rest of the tree. The nodes and edges of the tree are usually decorated in a particular notation in order to indicate the dependencies and constraints placed on the features. It is possible to let the model take on a more general graph structure, but a tree is usually seen as more useful as it indicates more clearly the distinction between the different partitions of features in the model, as well as the further levels of detail that are added when the user of the model makes feature choices.

The information that is visualized in the feature diagram could easily and conveniently be contained in some other format for processing and storage, however one of the very points of models is the visualization aspect. We will therefore
use feature diagrams to introduce some of the semantics that has been used in feature models.

There is a substantial amount of information available concerning the notation of feature diagrams, as well as what kind of information that is suitable to be supplemented in a feature model. For a more detailed comparison of notations, see [121]. The usefulness and necessity of the extensions made to the feature diagrams is debatable. While in some cases it certainly adds clarity and brevity to the notation, it is argued in [10] that they do not add any more expressiveness to the diagrams. This is of course not the case if the extension adds notations and semantics that is brand new and did not exist in the first place, such as the hint stereotype from [98].

**Feature Types**

FODA listed three types of features:

**Mandatory**  This type of features represent the common parts of a product, meaning that they are included in every configuration of a product where the parent of the feature is included. Mandatory features that are connected to the root concept, and the mandatory features of those features, form a core or stem in the feature model, which represents functionality that is always included in all configurations of the product modeled.

**Optional**  These features represent the variability of a product. Depending on the functionality needed in the configured product, a feature of this type may or may not be included, if the parent of the feature is included.

**Alternative features**  Out of a set of alternative features, only one can be included in a product configuration, provided the parent of the alternative set is included.

Each feature can either be a concrete feature that represents a concept that can be implemented and included in the product as a real function, or the feature could be an abstract feature. Abstract features, or pseudo-features, represent logical concepts that provide an abstraction or connection point for a group of features, which in turn implement the abstraction and make it concrete.

Generative Programming [22] adds the type OR-features, which is similar to the alternative features type, except any non-empty set of the features can be included in the configuration, provided that the parent feature is included. This can then be combined with the other semantic types of features into a variety of different groupings like optional alternative features, optional or-features and optional alternative or-features, in order to allow for more variation in the
2.2. Feature modeling

minimum and maximum number of features selectable for inclusion in the configuration.

There is also the concept of parameterized features, which if included are assigned or assume a value of some type. An example would be the sampling interval of a sensor, or the fuel consumption of an engine.

There have been many different suggestions for classifications and categories of features. The original FODA proposed four feature categories:

- Operating environments in which applications are used and operated.
- Capabilities of applications from an end-users perspective.
- Domain technology common to applications in a particular domain, exemplified by navigation algorithms from avionics.
- Implementation techniques based on design decisions.

No further motivation is given for this choice of feature classifications, other than that they make sense in many common cases. Based on taxonomies from user interface design, the capabilities class is further refined into functional features, operational features, and presentation features in [66].

Yet another set of categories, referenced in [22], proposed by Bailin, speaks of operational, interface, functional, performance, development methodology, design and implementation features. Riebisch et al. suggest a distinction of functional features expressing the way users interact with the product, interface features expressing the conformance to a standard or sub-system, and parameter features expressing enumerable, listable environmental or non-functional properties. The grounds of this classification is that the feature model should serve as a customer view, and that all other information should be captured in other models during design and implementation. The proposed classification therefore reflects the customers view on a product, and in what terms a customer considers a product’s functions and capabilities.

While one could argue about which classification that is the most useful, it is the domain of investigation and the application of the modeling technique that should guide which classification that is most suitable. There is no reason to use a classification scheme that does not make sense in the current context, and one could thus imagine many other ways of categorizing features in order to assist in the production and use of the feature model. All methodologies for feature and domain analysis recognize this very fact, and suggest that the categorization should be done based on the experiences in the domain and according to the intended use and context of the feature model.
Interestingly, no author makes a clear-cut argument for the usefulness and purpose of classifying features, nor makes any statements about the benefits that would come from using a classification scheme. There seems to be a general agreement among the authors mentioned above [66, 22], that the classification should serve as a view on the system from the perspective of some stakeholder. This idea on classification does not seem very well thought through though, since the classifications often do not appear particularly striking for any well-defined set of stakeholders. In other words, the classification schemes seem to be a relic from a requirements engineering work pattern. In their defense, classification schemes can be useful for some stakeholders such as designers and testers.

**Supplementary Information**

Czarnecki and Eisenecker also elaborate considerably on the use of supplementary information for each feature in the model, among others semantic description, rationale explaining why the feature is included and when the feature should be selected, stakeholders that have an interest in the feature, exemplar systems, constraints, and priority. To include all this information in a feature diagram is not appropriate, so it is instead maintained in a separate document accompanying the feature diagram. These feature attributes provide means for storing information in the model in order to widen the field of application for the feature models. While the purpose of modeling commonality and variability is still the focus of the model, additional information makes it possible for more stakeholders to use the model in more contexts, where variability of the product is a useful abstraction. The information could typically be realized in some form of product database.

We list some of the more common, and probably useful, entries here, mainly as mentioned in [22].

**Semantic description** Each feature should have a short description of its semantics.

**Rationale** An explanation of why the feature is included in the model, and annotation of when the feature should be selected and when it should not be selected.

**Stakeholders** Each feature should have an annotation, listing which users, developers, software components, customers, client programs, etc. have an interest in the feature.

**Exemplar systems** If possible, a feature could have a note on existing systems that implement this feature.
2.2. Feature modeling

**Constraints**  Constraints are hard dependencies between variable features that dictate which features are necessary in order to ensure functionality. A description can also contain recommendations or hints on features required, given the inclusion of a particular feature.

**Availability and binding sites/mode**  An availability site describes when, where, and to whom a feature is available. An available variable feature has to be bound before it can be used. A binding site is the site where a variable feature may be bound. A site model could consist of predefined times, user actions, contexts, etc. The binding mode describes if the binding of the feature is static, changeable, or dynamic. Binding site can serve as yet another way of classifying features in a model.

**Priority**  A feature can have a priority assigned to it in order to reflect its relevance to the project or software line.

Several other entries have been suggested. One is to allow a feature to be associated to a particular type, such as integer or string, which would allow the feature to assume a value. Fey et al. [39] talk of properties that are associated with features and that let the feature take on values in the same way. They go one step further though, and discuss the possibility of dependencies and interactions between properties, as well as between features. The interactions of the feature properties are described as some form of information flow between the features, and the setting of one property in a feature to a particular value would affect another feature through a “modify” relation.

Some of the entries in the supplementary information are visible in the feature diagram, such as the constraints information and dependencies. The reason to include it also in written or non-pictorial form is to make it possible to add complex interdependencies that do not lend itself to the graphical format.

**Relations and Constraints**  The organization of the features into a hierarchy often seems intuitive at first glance, but at closer examination the semantics of the hierarchical structure can be a bit confusing. The general idea of the hierarchy is to structure the features in such a way that moving down the tree, higher degrees of detail are achieved and more detailed decisions on design are made. FODA hints that the semantics of the hierarchy is “consist-of”, and in FORM there are three different kinds of semantics, namely “composed-of”, “generalization/specialization” and “implemented-by.”

Generative Programming states that the semantics of the hierarchy cannot be interpreted without considering the features related and their types. The lack
of semantics in the feature model is deliberate and Czarnecki and Eisenecker say that structural semantics should be placed in another model, more suitable for it, such as entity/relationship-models.

Apart from the relations that arise as a consequence of the use of mandatory and optional features in a feature hierarchy, features in a model can also have other interdependencies to one another, that are not part of the hierarchy structure. FODA originally described two types of feature interdependencies, namely “requires” and “mutually-exclusive-with.” These are hard constraints used to, for instance, indicate that a manual transmission in an automobile is mutually exclusive with an automatic transmission. Whereas these constraints are sufficient for most purposes, they do not offer much in terms of flexibility and there could be modeling constructs that cannot be accommodated using only these basic relations [98].

In [39], the authors describe the “provided-by” relation used to realize an abstract feature into a regular, implementable feature. They also rename the consists-of relation to “refine”, and also loosen the boundaries between hard and soft constraints. Riebisch notes that the semantic difference between the relation that make up the hierarchy and the require relation is quite small. Also, the “mutex-with” and the alternative relations are similar in semantics and constitute different ways of achieving similar constructions in the feature model. See Figure 2.4 for an example of a transition. Riebisch groups “is-a”/“part-of” along with requires into the hierarchy, based on the features that exist at the endpoints of the relation.

![Figure 2.4: The equivalent constructions of the mutex and requires relations. [98]](image-url)
Generative Programming introduces so called weak constraints, which can be used to indicate default dependencies for features, but can be overridden if necessary. Riebisch calls this particular type of relation for hint relation. Riebisch also adds the refinement stereotype, which is used to point to features that are significantly effected by the inclusion of a certain feature. Several other authors have added other kinds of constraints, relations and tracing mechanisms to feature models [22, 98, 38, 108].

In theory, one could adorn feature models with any amount of information concerning dependencies and tracing between features, but in practice the models tend to expand quite rapidly for anything but the simplest systems, even using the basic stereotypes. Using stereotypes is however a powerful means of expanding the semantics and abilities of the feature model, as it is in UML.

As previously mentioned, one use for feature models is to configure instances of product families, using reusable assets, components, and packages. In order to create an instance, there must be relations between the features selected and the corresponding assets and building blocks. Different approaches to feature modeling manage the connection between features and artifacts in different ways. In FORM the artifacts corresponding to the features can consist of pre-coded modules, skeletons or templates that are parameterized. Generative Programming brings up generic programming, template-based C++, aspect-oriented programming and generators among other techniques.

Notations of Feature Diagrams

Development of notations for feature diagrams has been a quite lively research topic, along with developing meta-models for feature models. Various authors have developed, extended or modified the notations and semantics of feature models, in order to cater for different needs that have occurred through the application of feature models in practice. We will not bring up feature modeling meta-models to any greater extent, as the notations for feature diagrams generally fully reflect the constructs in the underlying meta-model.

The original notation for feature diagrams comes from [66] and contains the basic building blocks of feature models, such as mandatory, optional and alternative features and some composition rules such as dependency and mutual exclusivity. Figure 2.5 illustrates the notation for mandatory, optional and alternative features, as well as some composition rules. As the constructs in this first version of feature models are quite simple and fundamental, the notation and appearance of the feature diagrams is also un-elaborative and easy to understand intuitively.
Figure 2.5: The FODA-notation of feature diagrams. [66]

Figure 2.5 also illustrates how supplementary information about composition rules and rationale is included as textual information alongside the feature diagram, rather than as edges in the diagram with stereotypes added, which we will see in later revisions of feature diagrams. Optional features are in this notation denoted by an empty circle, and groups of alternative features are denoted by an empty arc connecting the edges to the groups features.

The changes made in FORM, the successor of FODA, introduced four layers in the feature model denoted capability layer, operating environment layer, domain technology layer, and implementation technique layer. As seen in Figure 2.6, FORM uses the interdependency types generalization/specialization, composition and implementation, tracing these interdependencies across layers in the model. The resulting diagrams very quickly become hard to overview, but the problem is to some extent mitigated by not including the composition rules in the diagram. The decorations of the features are the same as those in FODA. FORM represents a substantial expansion of FODA, not so much in notation as in the application of the feature model, and the target of the model. By layering the information, the features are grouped in a way that illustrates decisions on design and implementation issues on different abstraction levels.

In Czarnecki and Eisenecker [22], the FODA-notation of feature diagrams is slightly modified and also extended to include OR-features. Figure 2.7 illustrates the notation used by Czarnecki and Eisenecker. Mandatory features are here decorated with a filled circle, while optional features have an empty circle. OR-features are indicated using a filled arc between the edges of the group of alternative features.

In [98], Riebisch et al. makes further extensions to feature models by adding more stereotypes to relations between features such as hints and refinement.
They also suggested changes in [99], with the use of multiplicities to denote the choices of features. Figure 2.8 illustrates some of the changes and additions.

The previous notations used the composition rule of feature/sub-feature and alternative features to implicitly indicate the relations requires and excludes, and complemented this with the use of textual information describing composition rules. In Riebisch’s notation, stereotypes are added to additional edges between features to indicate these types of dependencies. This makes it possible to model the features a bit more freely in the decomposition hierarchy, while not sacrificing the possibility to denote hard and soft constraints in the diagram. The choice of whether to include such constraints as objects in the diagram, or whether to use information in textual or other formats apart from the diagram, is up to the user and the tools that are available to facilitate, not only the construction of the model, but also the use and understandability of it.

A notation for parameterized features is also introduced in this variant of the feature diagram notation. By using multiplicities in the arcs of optional groups, the notation is clean up, and the need for OR-features and alternative feature
groups is eliminated. The use of multiplicities is similar for instance to UML, where a range on the form 2..* indicates that at least two features has to be selected, and the maximum number of features allowed to be selected is unbound up to the total number of features in the group.

**Feature Modeling Methodologies**

FODA and its successor FORM share most characteristics with each other and most subsequent methodologies for feature analysis are heavily based on FODA, which makes understanding of the FODA-process a prerequisite to understand most methodologies on the subject. A generic process for feature analysis consists of:

1. collecting information sources,
2. identifying features,
3. abstracting and classifying the features into a model,
4. defining the features,
5. validating the model.

Among the information sources used for finding features is documentation, such as user manuals, requirements documents, design specifications, implementation documentation, and source code. Apart from product documents, one can also use standards, textbook material, and also domain experts. When processing the sources for potential features one should take care to resolve ambiguity in meaning of concepts. Understanding the language of a domain is generally regarded as an effective way of finding possible features.
Once features are identified, they should be classified and structured into a hierarchical model using the consists-of relationship. During the modeling, each feature should be indicated as being mandatory, optional, alternative, etc. Each feature should also have resolved dependencies, and should be supplied with additional information. In order to ensure that the feature model is made as complete and useful as possible, it should contain features from both high levels of abstraction, such as functional and operational features, as well as more technical features representing implementation details. The structure of the feature hierarchy could be quite varied. For instance, one should consider whether two features in different parts of the feature model, which are mutually exclusive should instead be organized as neighboring alternative features.

Once the model has been completed, it should be validated against existing applications and by domain experts. Preferably the validation should be made by domain experts that were not part of the construction of the model, since they are less likely to be biased. It could also be useful to validate the model against at least one application that was not part of the material analyzed for the model. Further details on the methodology is found in [66, 62].
Czarnecki and Eisenecker provide a feature modeling process in [22], which identifies sources for features, discusses strategies for finding features, and the general steps taken in feature modeling.

As sources of features, they mention not only existing, but also potential stakeholders, domain information available from domain literature and domain experts, existing systems, and existing models. The existing models could be available object models, use-case models, and other models created during design and implementation phases of the software development. For identifying features, it is important to remember that anything that a user might want to control about a concept could be a feature. Czarnecki and Eisenecker therefore consider implementation techniques and other implementation issues as features to be considered.

The book suggests looking for features at all points in development and investigate more features than what are intended to be initially implemented, in order to provide space for some growth in the future. At some point in the process there should be a scoping activity where the features to be implemented are agreed on. At this point, the use of priorities for the features are important.

The book also describes a “micro-cycle” of feature modeling, which follows the standard workflow of identifying similarities between all instances of the products, record the differences between instances, that is variable features, and then organize them into diagrams. After this, the feature interaction analysis ensues, during which contradicting dependencies are resolved. This could lead to the discovery of feature interactions and combinations that were not discovered initially. This workflow is similar to that of requirements engineering.

FeatuRSEB [46] is a methodology that integrates features and the RSEB-method [56], where feature models complement other models by showing which functionality can be selected when engineering new systems. The feature model is used as the connecting model among several others that are constructed as part of the work. The feature model is developed in conjunction with the other models and is, step by step, extended and refined with more information.

Construction of feature models in FeatuRSEB starts from use-case models, where individual use-case models are merged into a common domain use-case model using variation points to model the variability in use-cases. After this an initial feature model is developed with functional features derived from the domain use-case model. Using the RSEB analysis model, the feature model is extended with architectural features relating to system structure and configuration. The final model to be developed alongside the feature model is the design model. The feature model is augmented with implementation features as a result of this modeling effort.
2.2.2 Feature modeling research

Much effort has gone into finding efficient ways of managing variability in software and software-intensive systems. After FODA suggested modeling features, it was picked up and extended in several versions and integrated with various methodologies. Other suggested solutions for variability management include, among others, variability use-cases [30], variation points [49] and variability spaces [95].

While variability modeling is generally accepted as a beneficial practice, the merits of feature models are questioned on a regular basis [7, 13]. However, there does not seem to be an alternative that gains momentum at the expense of feature modeling. The usefulness of extensions and modifications to the feature modeling notations are also criticized as often being merely superficial and not adding any new actual semantics [103].

A hot topic for a number of years has been reuse and modeling of requirements and problem space structures [124, 26, 57]. While the management of variability in solution artifacts has become more and more commonplace, there are still substantial efforts needed to facilitate variability exploitation in the problem space.

Scalability has been an issue in variability modeling for a long time, and while it has been assumed that tool support would mitigate this to some extent, there are still efforts made to solve some issues from a methodological perspective. Development procedures and processes are being tested and tried in various settings and domains [58, 90].

Although feature models are employed in quite large operations and organizations, tool support is somewhat scarce. During the first case study described in this thesis, a survey of then available variability modeling tools was conducted. While major tool vendors provide a variety of tools for almost every step in software development, feature modeling is mainly treated in tool suites for development of software product lines and software families, and those tools are generally made by smaller vendors, such as Pure::Variants and Gears. The first tools for feature modeling were developed by researchers and research groups as prototypes to illustrate some aspects of feature modeling. Most of those tools were made to demonstrate proof of concepts rather than to support practical use and are insufficient for most modern applications of feature models. Examples of tools from this category are Captain Feature, AmiEddi, and ASI DAL.

Tool support and visualization for feature model development continues to receive attention from researchers with most of the efforts being made under the development of tools for software product lines, e.g. [63].
2. Foundations of Feature Modeling and Quality

2.3 Software quality

There has been considerable effort put into the task of establishing, measuring and determining properties of software artifacts and how they affect software quality. Starting with concepts taken from other engineering branches, the work on software quality gained substantial attention with the arrival of methods and guidelines for metrics and measurable properties of software quality [44, 76, 9, 54, 126]. The initial efforts were focused on implemented software artifacts and primarily code-centric. There were several metrics proposed that were intended to put a number on the complexity and level of completeness of the finished software product. The ISO 9126 standard has been constructed based on the plethora of quality models that were developed. We therefore present the fundamentals of the standard here and let it represent also the essentials of the prior quality models.

ISO/IEC 9126 [54] is the result of the standardization work on the comprehensive quality models that were published in the 1970’s, the best known being McCall [76] and Boehm [9]. Although the quality models of the 70’s were used and extended to suit particular purposes and apparently were regarded as useful and providing added value, there were problems with compatibility and terminologies between different models.

The ISO 9126 standard aims to provide a framework that can be applied to all forms of software products. The elements (characteristics, sub-characteristics and metrics) of the model should be chosen and adapted to suit an intended “purpose of usage” of the software product under consideration.

ISO 9126 defines the quality of software products in several scopes and relates ISO 9126 to other standards for the different scopes. On the top level, ISO 9126 regards the life-cycle of the software product. The origin is the quality needed and requested by the users of the software product, called quality in use. These needs give rise to the external quality, which represents the quality properties of the software product that are visible to the user of the product. The external quality depends on the internal quality of the software product, which is why there are defined quality attributes and metrics for internal aspects of the software product. Finally, the process quality influence the internal quality. Having a good lifecycle process in place results in improved internal quality, which eventually leads to higher quality in use. Figure 2.9 shows an overview of the scopes in ISO 9126.

ISO 9126 specifies six characteristics and 27 sub-characteristics for internal and external quality in software products. Table 2.1 shows the hierarchy of (sub)characteristics. For each (sub)characteristic, the capability of the software is determined by a set of measurable quality attributes. A quality attribute is a mea-
surable property of the product and can influence several sub-characteristics. The quality attributes are however separated across internal and external quality. There are several types of scales that can be applied for the measures, defined for the metrics of each quality attribute. Depending on the environment of the system, and the type of data collected, the scales could be binary (satisfactory/unsatisfactory) or divided in several degrees (exceeding, target, minimal, unacceptable).

The internal quality is thus determined by the sum of the characteristic as seen from an internal view. The requirements for internal quality detail what levels of performance that are expected from the software product. The internal quality requirements affect design and implementation decisions and could include metrics that are not mentioned in the ISO 9126. Direct internal metrics might not be attainable in some circumstances or for some qualities, in which case indirect measures of the consequences of the quality can serve as indicators.

Quality in use is a measure of to what extent the users can achieve their goals in a particular environment. Like internal and external quality, there are characteristics for quality in use, see Table 2.2, which represent the important qualities of a product, as perceived by a user. There are however no sub-characteristics. Different metrics and qualities are of course seen as more or less important by different users. One of the general problems of determining appropriate qualities stated in ISO 9126 is to reach all users, and acquire their view on what qualities that are most important.

The main merit of the ISO 9126 is the fact that it is a standard and thus represents a common and widespread view on quality in software products. As a standard, it is natural to discuss software quality in terms of ISO 9126 and reaching a consensus based on the provisions of the standard. While its use is spreading, it is nonetheless a rather lengthy documentation that needs to be digested and integrated into the practices of organizations.

The standard is kept general on a conceptual level, but the measurements that are suggested for the software products are not generally applicable to software development models without adaptations. We can thus not use the work from ISO 9126, and simply plug into a quality model for feature models, but

![Figure 2.9: Quality in the life-cycle. [54]](image)
Table 2.1: Quality model for external and internal quality.

<table>
<thead>
<tr>
<th>Functionality</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Suitability</td>
<td></td>
</tr>
<tr>
<td>Accuracy</td>
<td></td>
</tr>
<tr>
<td>Interoperability</td>
<td></td>
</tr>
<tr>
<td>Security</td>
<td></td>
</tr>
<tr>
<td>Functionality compliance</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reliability</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Maturity</td>
<td></td>
</tr>
<tr>
<td>Fault tolerance</td>
<td></td>
</tr>
<tr>
<td>Recoverability</td>
<td></td>
</tr>
<tr>
<td>Reliability compliance</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Usability</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Understandability</td>
<td></td>
</tr>
<tr>
<td>Learnability</td>
<td></td>
</tr>
<tr>
<td>Operability</td>
<td></td>
</tr>
<tr>
<td>Attractiveness</td>
<td></td>
</tr>
<tr>
<td>Usability compliance</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Efficiency</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Time behavior</td>
<td></td>
</tr>
<tr>
<td>Resource utilization</td>
<td></td>
</tr>
<tr>
<td>Efficiency compliance</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maintainability</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Analyzability</td>
<td></td>
</tr>
<tr>
<td>Changeability</td>
<td></td>
</tr>
<tr>
<td>Stability</td>
<td></td>
</tr>
<tr>
<td>Testability</td>
<td></td>
</tr>
<tr>
<td>Maintainability compliance</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Portability</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Adaptability</td>
<td></td>
</tr>
<tr>
<td>Installability</td>
<td></td>
</tr>
<tr>
<td>Co-existence</td>
<td></td>
</tr>
<tr>
<td>Replaceability</td>
<td></td>
</tr>
<tr>
<td>Portability compliance</td>
<td></td>
</tr>
</tbody>
</table>
2.4 Quality of software models

Table 2.2: Quality model for quality in use.

<table>
<thead>
<tr>
<th>Quality in use</th>
<th>Effectiveness</th>
<th>Productivity</th>
<th>Safety</th>
<th>Satisfaction</th>
</tr>
</thead>
</table>

we can use some of the structure, ideas and core concepts for a model aimed at feature model quality.

2.4 Quality of software models

It is fairly straightforward to evaluate the quality of a physical artifact’s precision or performance, and it is reasonably easy to perform measurements on the performance, requirements fulfillment, and characteristics of a software product. Software models are different in that they are models — so a theoretical abstraction — of a tangible artifact, rather than the actual specified artifact. This means that during the craftsmanship years of software modeling, the only specification to evaluate the quality of a model against were personal expectations, estimations, and prior experience of what worked and what did not work. Instead of an objective, systematic process for evaluation, the evaluation process for models is more subjective and harder to formalize [81].

Trying to measure, evaluate, or estimate the quality of software products has by now become prevalent in most methodologies and is the subject of lots of research as well as standards and certifications. Despite the fact that software models have long been integral and important parts of software development projects, it has taken a while for the community to muster interest in the quality of models. However, quality work on models has a running start, thanks to the prospect of transferring concepts used on other software artifacts to models. Over the last years, workshops and conferences have gathered researchers interested in the matter, and more and more publications on the topic are appearing. We are still in the early stages of forming concepts and practices, though.

Quality of models should of course be of high interest, due to the spreading wave of transitioning from code-centric development to different approaches for model-based and model-driven development. For SMEs, this wave is progressing somewhat slower, due to the often very particular requirements posed on development procedures demanded by such units. The limited resources and the need to utilize existing investments in highly specialized intellectual property, is a
Table 2.3: Sample of qualities in models suggested from literature.

<table>
<thead>
<tr>
<th>Quality</th>
<th>Model type</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complexity</td>
<td>UML</td>
<td>[67]</td>
</tr>
<tr>
<td>Balance</td>
<td>UML</td>
<td>[67]</td>
</tr>
<tr>
<td>Esthetics</td>
<td>UML</td>
<td>[67]</td>
</tr>
<tr>
<td>Simplicity</td>
<td>Data models</td>
<td>[82]</td>
</tr>
<tr>
<td>Integration</td>
<td>Data models</td>
<td>[82]</td>
</tr>
<tr>
<td>Language</td>
<td>Dimension of ontologies</td>
<td>[112]</td>
</tr>
<tr>
<td>Navigability</td>
<td>General model</td>
<td>[89]</td>
</tr>
</tbody>
</table>

combination that makes it complex and less attractive to introduce model-driven approaches.

In order to reduce the risk of investing resources in modeling, but not getting a result that is satisfactory and usable, quality modeling, quality metrics for models and quality evaluations could be helpful. While not being a guarantee for successful modeling efforts, they could guide the efforts to emphasize the quality aspects that are particularly important for the organization.

Moody [81] provides a good overview of the current state and future directions of quality evaluation of conceptual models. Many of the concerns for conceptual models are also valid for other software models, since conceptual models is a very broad term that can encompass most descriptive models used in software development. In this section, we discuss some of the key issues and interesting references.

2.4.1 Measuring model quality

Even though the focus of quality models was subsequently widened to other software artifacts than code, it has taken quite a while for models to get attention. Starting with trying to quantify and measure properties of data models [82], the ideas of quality assurance for models are now spreading to other phases and modeling activities [89]. Quality of models is now a rich research area. A review of information systems literature alone found over 50 approaches for evaluating the quality of models [81].

Examples of qualities that are assessed for software models include such diverse characteristics as complexity, balance, and navigability. Table 2.3 shows just a small sample of qualities for models that have been suggested in literature by various authors.

Every approach to evaluate the quality of models brings up the importance of devising clear definitions, procedures and measurements for establishing the
2.4. Quality of software models

Table 2.4: Sample of metrics for qualities suggested by various quality models.

<table>
<thead>
<tr>
<th>Quality</th>
<th>Metric</th>
<th>Scale</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simplicity</td>
<td>Number of elements and relations</td>
<td>Number</td>
<td>[82]</td>
</tr>
<tr>
<td>Flexibility</td>
<td>Senior management rating</td>
<td>Assessment</td>
<td>[82]</td>
</tr>
<tr>
<td>Language</td>
<td>Has meta-classes</td>
<td>Boolean</td>
<td>[112]</td>
</tr>
</tbody>
</table>

quality of a model in a comparable and repeatable manner. The general criticism by authors of new quality models, vis-à-vis the ones already existing, is that the other models are not sufficiently detailed, defined and non-ambiguous to use in practice.

Despite the fact that many authors like to point out that there are few measurable or objective metrics defined for models, there are many suggestions for metrics that can be compiled from the different works. For example, general complexity has been suggested to be indicated by number of elements and relations, fan-in and fan-out, external connections, etc. Table 2.4 lists a small sample of metrics gathered from part of the literature. Metrics for models have mostly been looked at for conceptual data models (as in [43]) and UML-models (as in [67]).

In addition to the numerous suggestions of qualities based on the structure of ISO 9126, there are also other approaches to evaluating the quality in models. One suggestion from Lindland et al. [71] considers models in terms of syntactic quality, semantic quality and pragmatic quality. The suggested framework describes goals for each quality and means to reach those goals. For syntactic quality, the goal is syntactic correctness. For semantic quality, the goals are validity and completeness. For pragmatic quality, the goal is comprehension. A twist of the framework is that it introduces feasibility. This involves determining if the model has reached a satisfactory quality. If the model is not satisfactory, the goal to be pursued should be decided on and the most efficient means to reach the goal should be evaluated.

When it comes to variability models, the quality aspect considered in most literature on software product lines, is that of the products coming out of the product line. An often touted aspect of product lines is that the quality of the products increases when using a product line approach, since the quality of the constituent components increases. In software processes, quality in products is seen as emerging from having quality in the process and development activities. Taken together, it is therefore natural to assume that quality in the participating activities and models in software product lines should also be considered as indicators of the quality of the resulting products. Devising metrics for variability
on model level has begun to receive some attention, such as a basic complexity measure [73].

2.4.2 Evaluating model quality

While the above mentioned metrics and measurements try to establish some form of technical, objective, means of determining qualities in models and allow for comparisons between models, there are other approaches that address subjective evaluations of models.

Models with more complex semantics have also been the subject of evaluation approaches based on characterization of quality attributes. For example, there exists several methods for evaluation and comparison of ontologies based on metrics and properties of the ontology, such as [112]. These include frameworks for determining the suitability of an ontology for particular purposes, both using quantifiable and qualitative metrics and attributes. Several approaches are covered in [50].

In [88], a model and procedure for evaluating modeling languages is proposed. The designers of the model have applied it to UML and business process modeling languages. The author of this thesis applied it to two different approaches to model variability – feature models and variability spaces. The results of that evaluation is published in [114] and found that the evaluation model was applicable for variability modeling languages. The evaluation is, however, peripheral to the topic of this thesis, so we do not detail the results here.

There has been numerous experimental research efforts that compare models and modeling notations in order to determine which design principle, notation or modeling approach that provides the most understandable, easy to use and efficient models [110, 93]. The common procedure in these cases is to devise two models using different approaches, have test subjects use the model, then ask the users how they perceived the models, and thereby determine which model that was deemed more useful. The author performed an experiment after this style for feature models, more on this can be read in section 3.5.1.

In summary, the quality models, and the procedures and approaches for quality evaluation of models, are numerous and there is an active research effort around the quality of software related models. Little is done on the topic of feature model quality though, and the work in this thesis aims to fill some of that gap. While much of the work on model quality does not translate directly to feature models, there are elements and core concepts that can be used, once particulars of notations and languages are stripped from the previous approaches.
2.5 Chapter summary

This chapter has covered the essential foundations of feature modeling and quality in software and models. Software models have a wide range of applications and modeling can bring several benefits to a software development process. Feature modeling is an interesting and effective way of describing variability and commonality in a product family. Feature modeling is based on principles of domain engineering, where the goal is to develop software artifacts for families of products, instead of single products. There have been many different feature modeling methodologies suggested, and there are various definitions of what a feature is. There are also different notations for feature models and their visualization in feature diagrams.

Quality of software artifacts is described in several quality models, most notably the standard ISO 9126. Quality models divide the aspects on quality into quality attributes describing an effect of the artifact. Software models have been the subject of quality investigations as well. There are several proposals for measuring and evaluating the quality of software models, but none yet for feature models or other variability models. Examples of qualities for software models include simplicity, flexibility, correctness, and understandability.
2. Foundations of Feature Modeling and Quality
Throughout the work conducted and reported in this thesis, several research methods have been employed. This chapter describes the basic premise and rationale of the chosen approaches, and describes the limitations and potentials of the thesis work that is the result of those choices. The remainder of this chapter explains the general reasoning of the research conducted, and provides a list of common criticism of model quality research and how these issues are addressed in this work. Since the empirical material for this thesis has been collected using surveys and case studies, these methods are explained. Validation of results from software engineering research is discussed, and finally the process of the research work conducted is described.

3.1 A view on software engineering research

Software engineering is not a deterministic physical process, but is influenced by social processes and stochastic variables. Gravity works the same between any two software engineers, but modeling, designing, and implementing a solution to a software problem is dependent on complex interactions and contexts. Finding a strict cause and effect relationship in software engineering is hardly possible, since the engineering process is always influenced by the people performing the activities, the constraints of the organization and changing conditions in technology, etc. Therefore we cannot aim to find a theory (or “model”, if one prefers) of
software engineering practices that always holds, is suitable or the best practice in any context. But we can find ways of increasing the odds for the theory to be applicable in many different cases.

The general approach of reasoning in this thesis is inductive in nature, by which we try to find a theory that is the best fit of the given evidence observed, strengthening the theory as we go. Theories can consist of different properties and be used for a variety of purposes such as design, explanation, or testing. A theory can be modified to certain circumstances, be the basis for new theories, or propose a new phenomenon. The work described in this thesis takes several of these approaches and incorporates them.

We modify the theories on quality attributes and quality indicators from existing work, described in chapter 2, to adapt them into a form that is suitable for feature models. In doing so, we synthesize the quality attribute approach with the theories on how feature models are constructed and used. We make a claim of how context and goals affect the practices and results of modeling features. We use frameworks for designing case studies to find evidence for the theories proposed, and we test our theories to determine their validity. Testing is partly done using theoretical constructs and methods, however it has been important from the start to arrive at results that have value for practitioners, and that are supported and tested by empirical material.

Our attitude to research methodologies and approaches used in this thesis has been more inclusive than exclusive. The work has included elements from both quantitative empiricism and constructivist research. In our view, the approaches described in this chapter have worked complementary and improved the understanding of the concepts and relations detailed in this work.

3.2 Issues in model quality research

In [81], Moody provides an interesting list of theoretical and practical issues that have been found for model quality research. We recount the list here and provide a brief statement on how the work conducted over the course of this thesis has addressed these issues.

**Proliferation of proposals** While there are many proposals for evaluating the quality of UML- and data models, much less effort has gone into the quality of feature models. Since this work is the first of its kind, it is filling a gap.

**Lack of empirical testing** Out of the over 50 approaches reported in [81] only 20% were empirically validated. In this thesis, we account for several
3.2. Issues in model quality research

sources of empirical validation of our work. We have collected information from industrial practitioners using surveys and case studies. We have also investigated using controlled experiments for evaluating feature modeling notations. Empirical basis and testing have taken prominent roles in this work.

**Lack of adoption in practice** While the approaches and results of this thesis have not yet had a chance to influence and affect the way feature modeling is performed, it shows promise to be easily integrated in existing and widespread feature modeling methodologies, as described in sections 4.1 and 6.7. In the industrial case studies performed, the methods have proved to be of practical value.

**Different levels of generality** The approaches developed in this thesis are intended for feature models and have evolved with feature modeling in mind. There are other methods to model the variability of a software product line, but they are not considered in this thesis. Methods that are based only on theoretical work tend to be more general. Our work has considered practicality important, and therefore limited its generality to feature models.

**Lack of agreement on concepts and terminology** There are many terms used for similar concepts across various approaches to quality in models. In this work, we attempt to follow the terminology of established approaches, such as ISO 9126, where applicable. When there is a risk of confusion, we try to clarify and define the terms and the differences in terminology.

**Lack of consistency with related fields and standards** An assumption during this investigation of quality in feature models has been that the work can be based on, and related to, the existing work on software quality. Although many other proposals to quality in models have been constructed from purely theoretical ideas and chosen to not build on the existing work made in software quality (such as [71]), that is not the case of the work described in this thesis.

**Lack of measurement** Many methods are lacking in defining clear and consistent means to measure the quality of models. The reason is partly inherent to the nature of models, as mentioned in section 2.4. Feature models have both an advantage and disadvantage compared to other models. The advantage is that feature models often describe fairly tangible concepts, as the whole idea of feature models is to illustrate meaningful dif-
Lack of evaluation procedures  The procedures for quality evaluation, the processes and the application of the procedures are clearly defined in this thesis. The procedures are kept simple and straightforward, and their integration into the feature modeling workflow is explained. Descriptions of the instrumentation used is also provided.

Lack of guidelines for improvement  It is obvious that unless the quality can be improved, there is no point in making evaluations of it. We provide an initial set of development principles, based on the empirical material, that encourage the emergence of particular qualities.

Focus on static models  This issue from [81] concerns the lack of quality evaluation for dynamic models. Feature models are static models by nature, so this issue is only included here for completeness.

Focus on product quality  Product quality in this issue means the quality of the resulting model, while many other software quality mindsets also put focus on the process quality. It is assumed that the quality of the development process affects the quality of the resulting artifacts. In this thesis, little attention has been payed to the quality of the feature modeling process, although section 4.1 does look at how existing feature modeling methodologies consider quality of the resulting artifacts.

Lack of knowledge about practices  It was apparent from the start of this work that in order to provide means to improve the quality of feature models, it is essential to learn more about the needs of the targeted practitioners. To accommodate this necessity, we have gathered information both using surveys of current practices in variability modeling, and also learned of the practitioners’ needs from the case studies.

3.3  Survey methodology

Surveys are typically used when a researcher intends to form a view on frequencies and prominence of some variable of interest. Typically, the setting of
3.3. Survey methodology

<table>
<thead>
<tr>
<th>Method</th>
<th>Form of research question</th>
<th>Requires control of behavioral events?</th>
<th>Focuses on contemporary events?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experiment</td>
<td>How, why?</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Survey</td>
<td>Who, what, where, how many, how much?</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Archival analysis</td>
<td>Who, what, where, how many, how much?</td>
<td>No</td>
<td>Yes/No</td>
</tr>
<tr>
<td>History</td>
<td>How, why?</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Case study</td>
<td>How, why?</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

The variable is not possible to control or know in advance, in contrast to controlled experiments in laboratories (see Table 3.1 for a comparison of research techniques). The questions under investigation are of such types that they can meaningfully be posed to many respondents and the answers are of interest for a certain population. This means that the questions that can be investigated with surveys are often kept on a more general level than those that can be looked into using experiments or case studies.

Most commonly, surveys intend to predict or represent frequencies, or prominence, for a population based on asking a sample of the population. These kinds of surveys are often mainly quantitative and use questionnaires. There are also more qualitative approaches, commonly using interviews. In the first case, statistical significance and probabilistic methods are used to draw conclusions about the population based on the sample, while in the second case the data is analyzed in a qualitative approach [48].

A survey is an excellent way to acquire an overview of the current status of a variable in a population, provided that the selection of the sample is sound. In software engineering, surveys often have relatively low return ratios [106]. Most samples that end up being used are practically self-selected, which affects the quality of the survey if the responding sample is biased. The issue of low return
3. Research Approach and Process

ratios affects the quality if the topic under investigation is dependent on the reason for non-respondents not responding. There are examples where results from surveys about medical conditions have been biased because the sample affected by the medical condition was reluctant to share information about it, thus the surveys underestimated the prominence of the condition [40].

The underlying key statistic of the survey should not affect the response-propensity, which is hardly ever the case in surveys about software engineering matters. We have therefore used a survey as a means to form a requirements context for our work, finding current practices of variability management in SMEs, and collecting needs and demands (section 4.2) on efficient variability modeling from industrial practitioners.

The survey approach used has been implemented as questionnaires sent out to software developing organizations in the region around Jönköping University. The return rate is in line with that of many other reported surveys on software engineering [106]. Although a quantitative approach and analysis has been used, the survey results should not necessarily be regarded as having statistical prediction and generalization properties. The sampling has not been made based on a selection process that produce statistical significance forecasting. Instead, the survey should be seen as a “total survey” or census-like investigation, where the results are applicable only to the survey respondents and generalizable to, at best, the population that was surveyed. Statistical significance as such is not a quality measure of the survey, but only a probability of getting the same results regardless of what sample you choose from the population [79].

Many users of surveys fail to make the distinction between total surveys and surveys with statistical properties. In many of those cases, the return ratio and sampling process are perfectly inadequate for the predictions and generalizations claimed.

3.4 Case study methodology

While surveys are very suitable for finding practices on a general level among a wider population, the case study is suitable for asking questions on more detailed levels. Typically, a case study aims to answer questions on how or why about contemporary events, while surveys ask questions about post-event data, i.e. whether a phenomenon occurs. Case studies are in that respect similar to experiments, but unlike experiments, the researcher has little or no control over the context or environment that the observations are made in. That, in fact, means that case studies are the best choice for investigating research questions where the context of the observations is important and is predicted to influence the observed research variables. [128]
The term “case study” is fairly common in software engineering literature and research reports, but is very often used for, what should more appropriately be called, “experience report.” Case studies are usually seen as having low predictive and generalizing power. This is however a misconception and claiming that generalization cannot be made from a case study is like claiming that generalization cannot be made from experimentation\(^1\).

The mistake many make is to think that there should be statistical prediction capability in a case study, much the same as in a survey. However, a case study does not generalize to a population in the way a survey does. Instead, case study results generalize to a theoretical proposition [128]. Case study reasoning should be regarded as a strong form of hypothetico-deductive reasoning, not a weak form of statistical inference [100]. Cases are chosen for theoretical, rather than statistical, reasons. Cases can be chosen for replication, extension of theory, fill in categories of cases, or provide examples of polar types. [27]

Single cases are usually one of five types [128]:

1. Critical case, which – meeting all conditions for testing a theory – can confirm, challenge, or extend the theory.

2. Extreme case, which is a unique case and represent something rare, or interesting enough, to make it worth investigating and documenting.

3. Typical case, that captures the conditions and circumstances of an ordinary and commonplace situation, informative of an assumed average context.

4. Revelatory case, providing an opportunity to observe something that was previously inaccessible.

5. Longitudinal case, where a case is studied over two or more points in time.

In order to correctly draw conclusions from case studies we need an underlying theory with a proposition or hypothesis on the input and outcome of the case study. The results of the case study then provide us with evidence in favor of or contradicting the proposition(s) of the theory, allowing us to revise the theory accordingly. Obviously, just like with an experiment, the strength of the theory increases if the case study is replicated or repeated. The validation is similar to

\(^1\)For example, Galileo based his revision of the theory of gravity on one case, where the merit of the case lies in the choice of metal and feathers as extremes. By a carefully selected – critical – case, the validity of the theory was on good grounds assumed to hold for all objects in between the extremes.
traditional hypothesis testing, but each hypothesis is examined for each case, not the aggregate cases [27].

Research has shown that while the research community tends to put a lot of trust and scientific credibility into laboratory experiments, simulations, and theoretically validated results, industrial practitioners find more credibility in case studies and field studies. Case studies allow practitioners to evaluate the results of the conducted research in a setting and context, making it easier to relate the results to their own context and determine if adopting the proposed technology or methods would bring any benefits [129]. Many researchers are not accustomed to leaving generalization up to the practitioner, but it is commonplace in other fields, like law and clinical research [64].

The appropriate balance between on the one hand the research communities’ demands on formal rigor and scientific value added to the community knowledge base, and on the other hand the engineering communities’ demand for applicable and contextualized results is a tricky one. Where one community prefers results that can be generalized as far as possible, the other prefers special solutions for particular and precise situations. A properly set up and carried out case study can come a long way in catering to the needs of both those goals. There are several proposals for planning, conducting, and reporting case studies available in literature. Sections 4.3 and 5.2.3 describe the approaches taken in the case studies performed during the work on this thesis.

### 3.5 Experimentation in software engineering research

As researchers and practitioners have used variability modeling to describe the commonality and variability of product lines, several extensions and additions for the variability modeling notations have emerged, based on criticism of the previous notations with regard to usability, modeling power and understandability (e.g. criticism in [7, 13, 39]). However, there is little or no evidence in favor of these claims of increased benefits. Experimentation could serve to empirically identify the strengths and weaknesses of the variability models, and to attempt to connect characteristics of models to qualities, such as usability or understandability, in the models.

#### 3.5.1 An experiment in feature modeling

An experiment, with the goal to investigate which notation for variability modeling that provides better understandability and utility for users, was designed
3.5. Experimentation in software engineering research

The independent variable in the experiment was the modeling notation used in developing the feature models.

The dependent variable was model understandability, indicated through task accuracy and perceived level of understandability of the models by the subjects. As the claim of the proponents for revised notations of variability models is that they resolve matters of ambiguity and increase understandability, the spread of the answers to the tasks would indicate the ambiguity of the models, if we premise that ambiguity is the same as diversity of interpretation. A presupposition of the experiment was that if two models are information equivalent, so that the information in one model can be inferred from the other and vice versa, the model with the higher level of understandability results in higher accuracy in performing tasks using the model, or require less effort of the users in order to perform their tasks using it. We would thus be interested in looking for a correlation between the task accuracy and perceived ease of use.

Should no model show any considerable advantage over the other, we can suspect that understandability of variability models is more concerned with understanding of conceptual issues, rather than peculiarities in the syntax and notation used to devise the model. We are then interested in trying to discern whether the subjects understood the conceptual foundation of the model correctly, and also in trying to detect correlations between level of training and modeling experience and performance in the experiment. By empirically assessing the perceived value and understandability of variability models, we can evolve and further the notations and methodologies for using variability models.

The subjects, a class of master students in information technology, were given training material for the interpretation of the modeling notation used in their respective treatment, a model of a product line in the notation for their treatment, and a list of tasks. The subjects were asked to answer questions of whether or not the tasks listed could be performed using the models given. In order to correctly answer the questions, the subjects had to be able to understand the syntax of the diagrams sufficiently to analyze the aspects of the notations that were changed between the different styles. After completing the tasks, they were asked to answer some questions about their perceived understanding and their appreciation of the feature model’s notation, and their background knowledge in software engineering and modeling.

The experiment was carried out in an off-line lab environment, with 20 students on master level in information technology. Since modeling is largely an abstract and theoretical exercise, the ability to pick up a modeling notation is something that should not be significantly affected by experience of industrial application. Learning how to successfully utilize a modeling notation is a dif-
3. Research Approach and Process

The subjects were randomly assigned to one of two treatments and the assignment was balanced. In Generative Programming [22] by Czarnecki and Eisenecker, the original feature modeling notation was somewhat modified and extended in order to accommodate some constructs that were not available in FODA. Riebisch et al. described a new notation for feature modeling in [98], which claimed to increase the understandability compared to the notations based on the original FODA definitions. These two notations were compared in the experiment, using a product line describing calendar applications.

The results of the experiment indicated that the notation in fact did not affect the task performance or understanding of the models, but that the spread in answers was more dependent on experience and software modeling skills. The validity of the results was rather weak though. The experiences of the experiment indicated among other things that the subjects background and skill in software modeling has a considerable impact on the understanding of feature models. It is also evident that the size of the group of subjects is relevant in order to compensate for the spread in skills.

The experiment showed that the premise of the experiment design had potential. We might be able to use empirical experiments to show whether or not the modeling notations used are significant, or if it is in fact more relevant to focus on developing the conceptual foundations of variability modeling, in order to make use of the models easier.

However, the performance of the subjects on the experiment material was very poor with a large number of missing data points. The only tentative conclusion to be drawn from the experiment was that the ease of using feature models was more related to the general skill in software modeling than anything else, including notation. The experiment material would require significant modification and the experiment would need to be replicated with practitioners, making the course of action cumbersome. Experimentation was therefore not used for the rest of the work described in this thesis. In retrospect, we can speculate on a corroboration of the notation not being important, since the results from the evaluation of feature models points to prominent formal aspects, rather than usability (see section 5.2.12).

3.6 Validation in software engineering research

Validity is not a binary decision, it is a judgement of degree [64]. Neither is generalization a function of the number of units observed. More important are the kinds of units observed. Validity of research and empirical material is of-
Table 3.2: Research methods and their validity types. [127, 40, 128]

<table>
<thead>
<tr>
<th>Research method</th>
<th>Validity types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experiments</td>
<td>Conclusion validity</td>
</tr>
<tr>
<td></td>
<td>Internal validity</td>
</tr>
<tr>
<td></td>
<td>Construct validity</td>
</tr>
<tr>
<td></td>
<td>External validity</td>
</tr>
<tr>
<td>Surveys</td>
<td>Patterns of association</td>
</tr>
<tr>
<td></td>
<td>Alternate question forms</td>
</tr>
<tr>
<td></td>
<td>Validation against records</td>
</tr>
<tr>
<td></td>
<td>Consistency validity</td>
</tr>
<tr>
<td>Case studies</td>
<td>Construct validity</td>
</tr>
<tr>
<td></td>
<td>Internal validity</td>
</tr>
<tr>
<td></td>
<td>External validity</td>
</tr>
<tr>
<td></td>
<td>Reliability</td>
</tr>
</tbody>
</table>

ten divided into a number of validity types, for some reason almost always four. Table 3.2 provides an overview of validity types for the research methodologies discussed in this chapter. The validity of each contribution in this thesis is discussed in connection to the reported results, but here we mention some general aspects of validity that are applicable for the thesis as a whole, namely the validity of industrial usage in practice and observation of effect for research results.

It can be argued whether or not the “test of reality” holds any merit in academic research. Erdogmus touch on the issue in [28], where he lists a series of common objections against the use of empirical material and practitioner experiences and commentary on them. Table 3.3 lists three of those objections and comments that are applicable to the view on practice and investigation into practitioners’ habits.

Pfleeger [91] describes the process of technology transfer, i.e. the process of making practitioners adopt research results. With regard to technology evaluation, Pfleeger states that the first step of preliminary evaluation consists of an organization evaluating the benefits of the technology, relative to their current practices. If the results are useful in the particular organization, then we wish to know if the benefits extend to other similar or dissimilar organizations. The purpose of the initial evaluation is to produce evidence of the benefits – or drawbacks – of the new technology or technique. In order for this evidence to be of value for technology transfer, it must be read, understood, and believed by practitioners.
3. Research Approach and Process

Table 3.3: Objections to empiricism. [28]

<table>
<thead>
<tr>
<th>Objections</th>
<th>Commentary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Empirical research is too soft and does not produce technology.</td>
<td>Insights from empirical research drives innovation. Understanding how a development team works leads to better processes.</td>
</tr>
<tr>
<td>It is impossible to prove a technique efficient, there are too many factors.</td>
<td>The goal is to enable educated decisions in particular contexts, based on a persuasive level of understanding.</td>
</tr>
<tr>
<td>The evidence models used do not apply to software development.</td>
<td>They are not yet customized for software engineering as a practice, but they reveal underlying issues and provide a first approximation.</td>
</tr>
</tbody>
</table>

Drawing from the theory of diffusion of innovations, Pfleeger relates some key questions that the evidence should answer:

Relative advantage  To what degree is the new technology better than the existing?

Compatibility  To what degree is it consistent with existing values, experiences and needs?

Complexity  To what degree is it easy to understand and use?

Trialability  Can it be experimented with on a limited basis?

Observability  Are the results of using it visible to others?

Each of these questions are addressed in this thesis by:

- evaluating the existing approaches, and how they ensure quality in feature models,
- relating and basing the results in the existing work on quality models and quality evaluation,
- providing concise and straightforward guidelines and means for applying the techniques, as learned from the case studies,
- exemplifying the use of the techniques in practitioner-oriented setting,
• and providing a means to concretize the results of using the techniques in an evaluation procedure.

In this thesis we attribute substantial validation value to the use of results in practice. In the light of findings by prior research into technology transfer and the value of practitioner-oriented case studies, the applications and evaluations of the results in this thesis in industry contribute significantly to the validation from an engineering perspective.

3.7 Research process

An important element for most research efforts’ validity is the ability to repeat the research. This section summarize and show the major steps taken during the progression of this thesis work, in order to make transparent how the different parts of the work are connected. We also give some comments on the value and repeatability of the work that have contributed to the results in this thesis.

The general scheme of the work was to perform three phases; exploration and generation, refinement work, and results validation. Figure 3.1 shows an informal illustration of the order in which major work packages were performed and what phases they were in. Initial interviews in industry (not detailed in this thesis) motivated a wider survey in order to find the industrial needs and requirements on a quality model for feature models. Literature surveys and reviews set the frame of the first draft of the quality model, and an attempt at using experimentation. The initial quality model was used in the second case study, which resulted in the quality indicators and a refined quality model.

Figure 3.2 provides an illustration of how the research activities are related to resulting artifacts and publications. The illustration shows how the first case study, called SEMCO, contributed to the motivation of the work and also how the tool FMWIKIK was a result of the Tactel case study. The Tactel case study provided the validation for the resulting quality model and methodology for constructing and evaluating high quality feature models. The results of the research activities have continuously been published in conferences, workshops, and journals as the work has progressed.

The most obvious way to improve the quality of the research would have been to simply spend more time applying the models and methods, and thereby gather more evidence for supporting or rejecting the theories. As it turned out, the experiment approach was unreliable and did not provide enough material to warrant investing much effort into it. However, if the circumstances had been more favorable for experimentation, a series of trials could have provided interesting information that could have been carried into the case studies.
Figure 3.1: Illustration of thesis work ordered by time and phase.
3.7. Research process

Figure 3.2: Illustration of how research activities, results and publications are related.
3. Research Approach and Process

The interviews with industrial representatives were quite useful in designing the survey, but had to be cut short due to respondents changing employers. The interviews could therefore not provide sufficient material to be included as an empirical component in the thesis work. This was rather unfortunate, since the material from the interviews conducted was very rich and gave interesting teasers about the differences in needs between different domains. At the moment, that aspect is relatively absent in the thesis results.

There are two main alternative research approaches that we see could have been used instead of the approach that was used. The first would be a more theoretical and laboratory oriented approach. With regard to quality in software models, there are many proposals that have been suggested based primarily on theoretical validation, measurement theory, etc. (Recall that 80% of the approaches found by [81] were not empirically validated.) This approach would have the benefit of minimizing dependency of industrial partners for validation purposes, and could still have gotten quite far. However, since a premise of the work in this thesis was that the results should be useful for SMEs, there was a distinct need to test the results in SMEs. There is not sufficient information available regarding the practices of SMEs to be able to determine the applicability of results from a purely theoretical process.

The second alternative would be a version of the “industry as a laboratory”-approach [94]. Such an approach has the advantage of faster feedback, tighter evaluation of research artifacts, and closer proximity to the research problems. These cooperations between academia and industry are however longitudinal and need longer commitments from the involved parties. SMEs are often not able to make such long-term arrangements. Had there been practical opportunities for longer studies in industrial contexts, that would have been desirable, but the research process that was used cannot be said to be a significant cause for any shortcomings in the results.

3.8 Chapter summary

This chapter has covered the approach and process used for the research work reported in this thesis. Software engineering is a relatively volatile research field that has yet to find best practices when it comes to research methodology. Several shortcomings exist in current research concerning model quality and need to be considered. Surveys and case studies are chosen as the main research approaches in this work, since they provide empirical material with scientific value, as well as anchoring the findings in practice and applicability. An experimental approach was also attempted, but did not prove to be fruitful. Figure 3.2 shows how the activities and results of the research are connected.
Chapter 4

The Need for High Quality Feature Models

As previously mentioned, the application areas for feature models have increased over time, and there are now many scenarios that could make an organization consider developing a feature model. Many methods for feature modeling say that in order to make the most of the feature modeling efforts, it is important to do proper scoping of the domain covered by the model. But it is also important to consider the scenarios in which the model will be used and the restrictions and constraints that the model is subject to as a consequence of the scenario.

This chapter motivates the need to look into the quality of feature models. We start with a review of existing approaches to feature modeling and what means they have in place to raise the quality of feature models. The review provides a snapshot of the current practices with regard to feature model quality. We continue with an analysis and interpretation of empirical material from a survey of SMEs. From the analysis, we find the perceived shortcomings in industrial practice with regard to variability management. We also get a view of the needs that the surveyed SMEs have on variability management for their products, problems, and solutions. Experiences from an industrial case (SEMCO)
The Need for High Quality Feature Models

also provide material for formulating the need. Together, these inputs provide a motivation for trying to raise the quality in feature models, and the requirements that are posed on such an approach.

4.1 Current practice in feature model development

This section describes how current approaches to variability modeling address the issue of quality in the resulting models. A selection of methods, representing the current state of feature modeling methods, have their respective strengths and weaknesses concerning model quality assurance pointed out. These types of investigations are due to their nature notoriously hard to repeat with the same outcome. In order to perform an evaluation, one needs to determine criteria or baselines to evaluate against. In this review the criteria is based on the quality aspects found in other types of models or software artifacts, as recounted in section 2.4. The criteria are thus based on the best practices of quality assurance approaches to software products and models, but also take into consideration the concerns about variability modeling raised by small development units.

The feature modeling methods are inspected for five principal criteria:

1. reuser involvement,
2. scenario consideration,
3. feature model verification and validation,
4. quality attribute consideration,
5. meta-models and notations.

The criteria, selection of subject, and evaluation method used in this investigation are not formalized here, but the evaluation could be repeated for other methodologies.

4.1.1 A selection of feature modeling methodologies

The implementation of any modeling methodology is usually customized by the practitioner, therefore the presentation in this investigation is based on the available documentation and recommended best practices that have been disseminated to the community through literature and research reports.

There is a multitude of approaches for feature modeling available. Rather than looking at all of them, many of which are quite similar, a selection of representative approaches was made. Several aspects were considered when selecting
4.1. Current practice in feature model development

the approaches, e.g. proliferation, origin, notability, evidence of practical application, and availability of documentation. The methodologies that are reviewed here are a fair representation of the current state of variability modeling approaches.

FORM

Feature Oriented Reuse Method (FORM) \([70, 62]\) is the continuation of some of the earliest attempts at variability modeling that were significantly influential. It represents a relatively generic and quite recent approach to variability modeling.

In FORM, features are usually modeled into a hierarchical tree structure where features subsume sub-features and feature groups. Combination rules, such as AND/OR, are added to reflect constraints between features and feature groups. This is often visualized in a tree or graph with nodes and edges decorated with various symbols to denote the different types of features. An example of the graphical notation for FORM is showed in Figure 2.6.

Feature modeling in ESAPS

The lion's share of work in Europe concerning product line approaches has been conducted in the projects, sponsored by ITEA, called ESAPS, Café and Families \([36]\). The work packages of these projects has resulted in several methodologies and procedures for handling and modeling variability. Obviously, many of them are related and several have only been used in case studies within one participating organization and does not have much further documentation or public guidelines for usage.

Most of the methodological and theoretical underpinnings for variability analysis were carried out in the ESAPS project. The principal elements and concepts adopted are features and use cases. ESAPS Feature Analysis is described in \([4, 102]\) and presents the approaches used in the project’s case studies. The methodologies were mostly used in case studies as part of larger projects, where variability analysis and modeling were just one part. In addition to the mechanisms of FODA, ESAPS uses feature areas, grouping high-level features with as little inter-dependencies as possible, and relationships between those.

Each case study used its own format for representing the variability models, mainly stereotyping in UML and tabular based notations. The general approach of producing the models is to stepwise refine the structure of an initial list of high-level features into more detailed features and relationships.
Product Line Use case modeling for Systems and Software Modeling

Apart from describing variability in terms of features, there are approaches using use cases with extensions to describe possible scenarios of using the product line. Although use case scenarios has proven to be an interesting and useful technique to elicit requirements, they do not always fit well to variability. This has resulted in combinations of using feature models and use cases to provide a more complete modeling view on variability. One such approach is FeatuRSEB [46]. Based on RSEB (Reuse-driven Software Engineering Business) [56], it is an approach were feature models are used as a central model to connect other object-oriented models in a model-driven product line approach.

A further development and follower of FeatuRSEB is Product Line Use case modeling for Systems and Software Modeling (PLUSS) [29], which is intended for long-lived software-intensive systems, for instance defense applications. The goal of the approach is to provide a complete use case model for the system, while retaining a good view of the variability of the system. This is done by connecting use cases, features and use case realizations. The feature model is used to extract views on a use case that covers a whole program family. Each use case, scenario, or step is in turn connected to a use case realization showing which design elements that are affected by the use case, see Figure 4.1.

The capabilities of PLUSS are similar to those of other feature modeling methodologies. It has the same set of mandatory, optional, and alternative groups of features, but uses a slightly different terminology and graphical notation. It also supports the same set of restrictions and logical constructs, and in addition parameterized features. The available documentation of the methodology does not detail many guidelines or instructions for developing the variability models, but it does provide a notation to use and list feature constructions that are not suitable and should be avoided in PLUSS.

The approach has been applied in several case studies [30, 31] in an organization that produce defense material, reportedly with good results when compared to the previous approaches used in the organization.

SEI’s Product Line Analysis

The Software Engineering Institute, who are strong proponents of software product lines, also provide instructions and guidelines for Product Line Analysis (PLA) [15, 14]. The purpose of PLA is to build a requirements model for a product line of software intensive systems. This is partly done by using either a feature-driven or use case-driven strategy for analysis of the variability of the product line and its requirements, how the requirements of the products in the product line are affected by the variability, and how the requirements are con-
4.1. Current practice in feature model development

Figure 4.1: An example of PLUSS, showing the relations between features, use cases and scenarios. Each feature is associated with a use case and depending on feature selections, different scenarios come into play. [34]

This is modeled in feature models, use cases, object models that contain state and behavior, and a dictionary that contains the terminology used in the models.

The analysis guidelines describe both a feature oriented and a use case oriented strategy to model the requirements of a product line. The processes are iterative in their nature, like most modern requirements elicitation techniques and methodologies. Depending on which strategy that is used, the feature model, use case model, object model, and dictionary are seeded with terms and elements and are then refined iteratively.

The approaches described have been used in several industrial cases. Software Product Lines is currently the most prominent approach for large-scale
reuse in software development, and the product line analysis is an integral part of the practices for product lines, thereby still being actual.

**Organization Domain Modeling**

The Organization Domain Modeling (ODM) [105] method was developed under the auspice of the Department of Defense sponsored project STARS (Software Technology for Adaptable, Reliable Systems). ODM intended to provide practitioners of domain engineering with formal and detailed guidelines for defining, planning, and setting up domain engineering projects. ODM was designed to be a customizable and adaptable process model for domain engineering that can be used in various domains and integrated with various software development practices.

ODM makes a distinction between concepts, that are elements in the domain of interest, and features that are properties of the concepts. The methodology makes a point of keeping the concepts and features of a domain separate, so that features do not turn into concepts. The ODM notion of features is not focused towards end-user operations like those of FODA. Instead, features provide differentiation of key concepts. Therefore, features and concepts can be modeled in the same representation, and there is no need for a particular notation for feature models.

ODM is documented in relatively comprehensive publications, and while it is starting to grow a bit old many modern approaches to domain engineering and software reuse approaches still rely on and reference ODM as an example of an elaborate domain engineering processes. ODM was used in several projects and pilots under the STARS program, and also served as a starting point for domain engineering in Hewlett Packard.

**4.1.2 Criterion 1: Reuser involvement**

The reviewed approaches generally position themselves as stakeholder-driven during variability modeling, but the stakeholders considered are almost always the end-users of the product line. From a model quality perspective, it is also interesting to see what involvement the users of the modeled variability, i.e. application engineers or “reusers”, have.

Practically all methodologies for variability modeling include developers, application engineers and domain experts as stakeholders explicitly, and those that do not do so explicitly, do it implicitly. The main reason though is to provide input on the design and implementation of the assets or architecture for the resulting product line. Only in a few cases are they included to ensure that the resulting models and architectures are designed for use by them.
The PLA-approach explicitly discuss stakeholder involvement for checking domain- and usage appropriateness, but also place some emphasis on the development procedures, and have the users of the resulting models as stakeholders. The methodology also includes “development features”, which are features that control or influence the actual production of the products in the product line. PLUSS, which is relatively developer-oriented, also explicitly include the developers of the product line in deciding on the formation of the variability. RSEB, the basis of PLUSS also list reusers as stakeholders for refinement of the variability structure.

4.1.3 Criterion 2: Scenario consideration

A common element of quality assurance for software products is goal selection, where the stakeholders of the product get to negotiate and select which non-functional requirements on the product that are regarded as the most important to accommodate. The development of the product is then directed and governed by the priorities to ensure that the result meets the quality demands posed. This prioritization approach has been used quite extensively to guide the resulting products of software development, but less to guide the development activities. In the latter case, the stakeholders involved in the prioritization would be the developers of the product, rather than the users of the results from the development activities.

In the case of software architectures, there are examples of how the properties and behavior of the resulting product line architecture are affected by designing the architecture with particular quality constraints. By using scenarios to describe situations and uses of the architecture, stakeholders can prioritize the most important qualities of the architecture. This is followed by creating a quality attribute utility tree that refines and details general quality attributes to a concrete level, where there are measurable properties of the system, which the architecture has to support [16, 6].

Generally, the methodologies for variability modeling are directed towards producing models used for product line approaches. As stated in the introduction, small organizations can often have other incentives for doing variability analysis on their products ranges, at least initially. A variability model, produced for a different purpose than to serve a product line, could of course need other properties and other means of development. This would only be catered to if the methodology considers the intended usage scenarios of the resulting models.

All of the methodologies represented here have some level of consideration of the context in which the results will be used. For instance, PLA and PLUSS have a step where the stakeholders involved in the modeling are determining the
applicability and usefulness of the results given their particular point of view on the models. FORM and ESAPS’ feature analysis involve activities to ensure that the variability can be implemented in the product line, and that the variability is refined, so that it is aligned with the product line approach.

A model and a modeling approach is obviously much easier to use if they are included in the existing processes and development practices of the organization. Most methodologies for domain engineering and variability modeling have proposed a major turnaround of existing development processes, sometimes in a revolutionizing manner. For many smaller organizations, it is not always feasible to make such changes. With regard to the models that the methodologies produce, it is also interesting to ensure that they fit into the existing modeling practices and caters to utilizing the existing modeling artifacts.

Of the illustrating example methodologies in this article, it is only PLUSS that considers the existing process in the organization. Since PLUSS was developed from within the organization it was applied in, this is natural, but also means that the consideration is tailored to that particular organization. This in turn means that it could be challenging to transfer the method to another organization.

4.1.4 Criterion 3: Feature model validation and verification

Most things, variability models included, can in some way be validated and/or verified. In methodologies that use features as a primary element to model variability, the feature interaction problem is often brought up in conjunction with validation and verification. Feature interactions are unwanted or unpredicted behaviors that arise because of how features included in a particular configuration interfere and affect the system.

Several of the approaches to variability modeling discuss ensuring validity for variability models by using some form of logic or rule checker on the constructs provided by the methodology. The term verification is often more appropriate for this activity. For instance, FORM mentions a “constraint checking algorithm” to ensure that the constructs of the variability model are in order. In all cases, these validation efforts concern the technical quality of the variability structure, such as checking that two features that are mandatory are not also mutually exclusive. This type of check is often referred to as consistency checking the model.

Like most other approaches to domain and variability modeling, there is an iterative process in ODM. This is intended to ensure that the produced models are complete and correct, in the sense that all the important distinctions and dif-
4.1. Current practice in feature model development

All reviewed methods also include checking with stakeholders that the models correspond to the stakeholders’ wishes. This would represent validation, i.e. finding out if the model lives up to expectations. In most these cases, the modeled product line is what is up for evaluation, as opposed to the feature model. It is thus checked if the product line fulfills the requirements posed by the product line stakeholder, rather than the requirements posed on the feature model by the intended reusers.

FORM suggests instantiating the product line for each application considered in the domain analysis. PLA also recognizes that to fully ensure that all feature interactions problems are detected, it is necessary to test and analyze all possible feature combinations, which is unrealistic for variability models of a reasonable size. It is instead suggested that the investigated feature combinations should be selected on the basis on what qualities that are most important for the product line.

Determining whether models are appropriately designed for reusers is less attended to, but not entirely. FORM, PLA, and Generative Programming has recommendations on structural properties of variability models, such as rearrangement of features to align feature partitions with implementation modules, reduce complexity by moving common features upward in feature hierarchies, and resolve complex constraints by re-arrangement of features, see Figure 4.2 for an example. ODM brings up evaluation of certain properties of the model, such as well-formedness and style, suggesting that tools could be used for this, though not actually providing one.

Tool support for variability modeling is notoriously inaccessible. Apart from a small selection of commercial tools, there is a number of tools from research groups that are often developed solely to demonstrate a particular nuance of the group’s notation or methodology. PLUSS extends tools and workflows already present in the organization to accommodate variability modeling [33].

4.1.5 Criterion 4: Quality attributes considered

As seen in section 2.3, many suggestions for quality attributes have been made. The reviewed feature modeling approaches mention some quality attributes, all be it possibly redefined. Among the quality attributes mentioned are usability, completeness, parsimony, intuitiveness, accuracy, and correctness.

Rearrangements of model elements are attributed to increasing the usability of the model. Many methods regard usability as an important issue for variability
models, but only a few mean usability from the perspective of the users of the model, rather than the users of the product line.

Since, in many cases, a variability model is intended to capture some representation of requirements, the quality attribute completeness is often brought up as an important quality of models. Completeness is usually treated the same way as validation, in that stakeholders instantiate a selection of possible products from the product line, and thereby identify missing functionality from the variability model.

ODM has a pervasive, continuously running, validation activity that mentions several other quality attributes of the models produced using the methodology, among them minimality or parsimony. Keeping the models at a reasonable size and being economical with the model elements is seen as a means to achieve
4.1. Current practice in feature model development

intuitiveness. Intuitiveness is how well the model lends itself to being learned, if the model is natural or contrived, and if it can be internationalized in a readily manner. ODM also mentions model balance, meaning that there should be consistent and comparable levels of detail throughout the model structure.

The documentation of PLA actually mentions quality aspects of the resulting models. For usability, they should be easy to navigate, communicate to stakeholders, and update. For usefulness, a model should present accurate and consistent information that reflects the views of the stakeholders. The model quality activity should ensure that the structure of the models supports reuse, and that the relations between elements in the different models are consistent. The explicitly mentioned activities are ensuring that the models are understandable, and that they are able to adapt to future changes. There are no details or guides on how to actually perform these activities though.

A commonality of current approaches is the consideration of size and complexity for variability modeling efforts. Since the complexity of variability models increases rapidly with the addition of variable elements, the combinatorial explosion of possible configurations of the modeled domain is an early recognized problem. The generic answer to the threat of exploding complexity is to introduce tool support.

Since domain engineering methods are normally positioned as most useful in larger organizations and used for large projects, the chance of ending up with large models is of course seen as substantial. All methods acknowledge potential problems, but few suggest consequences or appropriate measures for mitigating the risks.

What is missing from all methodologies’ concern for quality of the models are some form of measurement instructions, indicators, or metrics definitions. While in most cases there are advice on making sure the models are comprehensible and well-formed, there are few pointers on what properties that actually make a model well-formed. Most methods prescribe using tool support or automation for evaluating the structure of the models, but there are no limits, levels, or constraints specified that can be used to determine under what conditions the models are poorly structured or of too high complexity.

4.1.6 Criterion 5: Meta-models and notations

Almost all methodologies add their own notation, or expand on existing notations, in order to support any novel concepts introduced by the methodology. Generally, this makes it easy enough to utilize the modeling approach, as all approaches properly cover the appropriate semantic constructs that are necessary for the uses described in the methodologies. In a few cases, the notation
is specialized for a particular purpose or domain, which increases the usability of the methodology for some particular purposes, but reduces generality. An example would be how some methodologies propose categorization and classifications of features that are distinctly aimed towards a particular domain, such as telecommunication or automotive.

New notations are often proposed with the argument of catering to some missing aspect or construct of the existing notations. In some cases, the notations are presented as being easier or more intuitive to use than previous notations, although there is no actual evidence presented to support such claims. The new concepts in the modeling notations are often related to adding some form of functional ability, rather than considering usability of the notation.

4.1.7 Summary

Table 4.1 shows a summary of the evaluation criteria and the consideration taken by each of the reviewed approaches in this section. Where an approach shows adequate consideration, the intersection with a criterion has is a filled symbol (●) and where there is directly inadequate consideration there is an empty symbol (○). If it is unclear how an approach is positioned on a criteria, the space is left blank.

While the reviewed approaches demonstrate fully adequate consideration for providing users with proper modeling constructs, there are limitations in making sure the produced models actually live up to the expectations and needs of the reusers.

To remedy this situation one can base modeling efforts on the best practices that has been found from experience, but for inexperienced organizations the training effort can be yet another hurdle to overcome in order to reap benefits of variability modeling. Developing defined activities and means for directing modeling efforts based on best practices, would be the first step towards a solution. Making these activities and means not only fit existing modeling approaches, but also existing development processes of SMEs, would ease the introduction of variability modeling in smaller contexts. One means to efficiently direct modeling efforts would be to provide prospect modeling organizations with scenario-based guidelines for modeling which, to a greater extent than what is the case today, takes into consideration the goals, context and available resources of the organization.

Looking at how quality assurance is conducted for other types of software engineering products and models, and contrasting that to the practice of variability modeling, one can see that there are some commonly practiced steps

\footnote{Variations of this notation will be used also in other tables throughout the rest of the thesis.}
4.2 Survey on industrial state and needs

We continue our motivation for high quality feature models by looking at industrial state and needs, regardless of method used.

This section presents some results from a survey that was conducted with organizations in a region in southern Sweden that is rich in SMEs. Particular attention is paid to the current use, practice, and perceived potential of variability management for software-intensive products and services. Over time, most organizations tend to develop a growing range of different products, services and offerings, while retaining legacy products and services [78]. A growing SME, that finds new market opportunities and wishes to efficiently utilize the existing assets, is likely to find itself in need of restructuring and refactoring of the lines of products and services offered. For instance, Hofer [52] makes an interesting comparison between very small enterprises and small enterprises, indicating that the need for more efficient and systematic approaches to software develop-
4. The Need for High Quality Feature Models

The need for high quality feature models appears quickly, with even moderate increase in organization size. With the increased complexity of software development added to the mix, having efficient and appropriate variability management in place is of high interest for such organizations.

The results presented in this section aim to describe the current state and practice of variability management for products and product line components in SMEs involved in development of software-intensive products or services. The main contribution of the survey is to let empirical material illustrate the situation and differences in needs between SMEs and the organizational types that are considered in much of the other variability research. Hopefully, the characteristics of variability in SMEs presented here contribute to making future research in variability management consider additional aspects of small-scale software developing organizations. This section provides a broader overview of potential and practice in SMEs, while most other work has been more of experience reports and case studies.

Variability management does not only concern code parts and implementations, but also other software development artifacts like product requirements and generalized domain requirements [83], models [20], designs, documentation and test cases [77], etc. The results from the survey are presented divided into three parts concerning three aspects of software development for a variety of products, namely (i) product structure, (ii) problem space structure, and (iii) solution space structure. Product structure concerns the range of actual end-user products or services that are offered towards markets. The problem space represents the requirements and constraints that are put forward by customers, but also standards, regulations and other sources that mandate how the product is constructed. The solution space contains the core assets, the solutions, and artifacts that are available for the organization to reuse. These aspects are of course tightly interdependent when it comes to handling the variability and commonality in them, and a key to effective management of a product line is to have a model and process in place to coordinate and utilize the possibilities given by the product line.

We note some key figures and trends, but mostly focus on the descriptive statistics. Though we do not try to draw any statistically significant conclusions, we can still illustrate how the situation is in the surveyed parts of industry with regard to variability management. The survey was distributed in Swedish and the questions and answers have here been translated to English.

The questions in the survey covered many aspects of software development practices and they contain more information than is disseminated in this thesis. Other aspects found in the survey have been reported in [118, 119]. This pre-
4.2. Survey on industrial state and needs

sentation concentrates on the questions concerning variability management for the organizations’ offered software products.

4.2.1 Survey participants

The survey was sent to software developing organizations in the region of southern Sweden where Jönköping University is located. The region is traditionally rich in SMEs and manufacturing industries. The organizations that were targeted for the survey were drawn from public registers of companies that had registered activities in the software sector. In the initial circulation of the survey, approximately 260 organizations were approached. As expected, some organizations replied that they were misrepresented in the registers and did not develop software, some organizations had restructured or ceased operations, etc. The final return rate of the survey was 13%, which is comparable to most software engineering surveys, although [106] claims to have found a consistent response rate as low as 5%.

For this survey, 25 responses are considered, all of which falls under the definition of SMEs as put forward by the European Union [113], meaning that they have less than 250 employees. Two additional responses were received from organizations classified as SMEs, but since they do not produce software products of their own, they do not exhibit variation in the product segment.

It could be noted that finding small-scale software developing units is far from trivial. While SMEs are easy to identify on size, we also find software development being conducted at a small scale in larger organizations. An organization of 1 000+ employees that develop products of reasonable complexity could have software components in the products that are developed by a software unit of perhaps a dozen staff members. Locating such units is quite a challenge. An additional, possibly confounding, factor is that the size of the organization in number of employees does not necessarily reflect the number of employees involved in software development. The actual resources available for the software development organization is thus often hard to estimate.

4.2.2 Instrumentation

The validity of an instrument – in this case a questionnaire – is not a measure on the instrument itself, but the notion of how justified the interpretation of the instrument scores are. The survey that was sent out was not solely concerned with the aspects of software development that are reported in this section. However, we did not see any tendencies in the replies that this introduced problems. Surveys typically have lower validity at the expense of higher reliability, since the
4. The Need for High Quality Feature Models

<table>
<thead>
<tr>
<th>How much influence do customers have over what tools you use?</th>
</tr>
</thead>
<tbody>
<tr>
<td>○ None</td>
</tr>
<tr>
<td>○ Marginal</td>
</tr>
<tr>
<td>○ They can make propositions</td>
</tr>
<tr>
<td>○ They are asked for suggestions</td>
</tr>
<tr>
<td>○ They have great influence</td>
</tr>
<tr>
<td>○ They can control what tools that are used</td>
</tr>
</tbody>
</table>

**Figure 4.3:** Example of survey question and alternatives.

Questions in a questionnaire survey are less rich in the information they provide compared with interviews. Repeating a survey is on the other hand easier.

The questions were constructed to let the participants choose the alternatives that best describe their situation and current practices, meaning that most questions were multiple choice questions. The sum of some answers’ alternatives can thus be higher than the number of participants. Because of the difficulty of predicting the response rate to the survey, the questions were not designed for extensive statistical analysis.

Only a few of the questions were of a Likert-scale type, and those questions did not use numbers. Instead, they had terms for each scale step that were specific for that question, see Figure 4.3 for an example. Phrasing the alternatives like this makes it easier for the respondents to relate the questions to their actual practices, rather than assigning a more abstract number to their situation [65].

We asked questions on the views and practices of the respondents according to product structure, problem space structure, and solution space structure. The terms product, problem and solution were explained in the questionnaire’s introduction. The respondents were encouraged to relate the definitions to their context. Different roles in an organization would encounter different types of problems and produce various forms of solutions, so like [84], we kept the terms in the questions general and exemplified them for different roles in the introduction.

The questionnaire was tested and slightly revised before distribution.
4.2.3 Validity and appropriateness

External validity is usually thought of as the ability to generalize results of an empirical material and its analysis to outside of the investigation's settings [127]. It is important to realize the difference between a statistical survey and an exploratory survey. When we approached the organizations on participating in the survey, we did not attempt to draw a representative sample of the software developing SMEs in the target region. This would have been practically impossible, since it is not feasible to characterize all of the variables and properties of all the organizations in order to make a representative sample. We sent the survey to all the organizations, meaning that we in effect tried to perform a sort of census on the entire population, rather than looking at a sample and then attempt to generalize to the population. This is a very common approach in many other surveys, yet the analysis of the respondents is still treated with statistical methods that are intended to generalize a carefully selected sample to a population. Miller [79] has described many of the pitfalls of applying statistical methods to empirical studies in software engineering, saying that the rush for adopting a “scientific method” in software engineering research has led to statistical significance testing being dominant, but unsuitable.

The consequences for external validity of this type of survey are that the results cannot be quantitatively and statistically generalized to other settings or populations, but there can be a qualitative argument made. In the presentation of the results when we point to trends, indications and correlations, we can only reliably say that they hold for the respondents. We can argue that the figures that we find are likely to be applicable also to the entire surveyed population, based on how the respondents correspond to the purposes of the analysis set forth in this report, i.e. we have a reasonable distribution over domains, organization size, maturity, etc. We can however not say that the results are likely to be applicable to other regions or countries, since the region we have surveyed is characterized by SMEs and cooperations between SMEs in a manner that may not be found in, for instance, a smaller metropolitan region, dominated by fewer, large operators.

The main factor affecting the internal validity of our survey is the fact that the respondents are self-selected. Self-selection introduces the risk of getting bias in the answers because of unknown or uncontrolled factors that influence the propensity to reply to the survey or influence the answers given. However, surveying software engineering practices could be considered relatively immune to most factors that might introduce bias since the topic – and subsequently the questions posed around the topic – are relatively free of reluctance to share information about it.
There is always a risk that a survey that is broadly distributed does not end up in the right hands in an organization, and that the answers received are not representative for the organization. In the case of SMEs, this threat is smaller, since the respondent in a smaller organization is likely to have better overview of the complete operations and practices, than what would be the case in larger organizations.

### 4.2.4 Respondents’ characteristics

The respondents of the survey are all working in SMEs, with a majority representing organizations with less than 51 employees (Figure 4.4). The characteristics of the software produced range from being used in embedded systems to user interfaces. Figure 4.5 illustrates how software is used in the respondents’ products to provide various functions. The organizations conduct their business in a wide variety of domains, including defense, telecommunications, business systems, and most prominently, consultancy for Internet and web solutions as seen in Table 4.2. As expected in small to medium-sized organizations, the respondents fulfill several different organizational roles in their respective organizations, showed in Figure 4.6. A majority of the organizations have a typical run-time of less than six months for a development project.
4.2. Survey on industrial state and needs

Figure 4.5: Use of software in the respondents’ products.

Figure 4.6: Organization roles of respondents.
4. The Need for High Quality Feature Models

Table 4.2: Number of respondents active in various domains.

<table>
<thead>
<tr>
<th>Domain</th>
<th>Respondents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defense</td>
<td>4</td>
</tr>
<tr>
<td>Telecom</td>
<td>6</td>
</tr>
<tr>
<td>Medical systems</td>
<td>1</td>
</tr>
<tr>
<td>Manufacturing</td>
<td>7</td>
</tr>
<tr>
<td>IT-consulting</td>
<td>10</td>
</tr>
<tr>
<td>Aerospace</td>
<td>1</td>
</tr>
<tr>
<td>Vehicle</td>
<td>1</td>
</tr>
<tr>
<td>Research</td>
<td>2</td>
</tr>
<tr>
<td>Security</td>
<td>5</td>
</tr>
<tr>
<td>Business systems</td>
<td>8</td>
</tr>
<tr>
<td>Web/Internet</td>
<td>10</td>
</tr>
<tr>
<td>Education/Training</td>
<td>4</td>
</tr>
<tr>
<td>Government</td>
<td>4</td>
</tr>
<tr>
<td>Tools/support</td>
<td>6</td>
</tr>
</tbody>
</table>

4.2.5 Product variability

The predominant method of describing and documenting the variability in the organizations’ product structure is natural text. While the respondents represent various ways of modeling the product structure, illustrated in Figure 4.7, it can be noted that all organizations with 25 or more employees have some sort of documentation of the product structure, and the formality level of documentation increases as the organization becomes larger. The organizations with 51 or more employees are all using architecture and component descriptions.

Table 4.3 shows the documentation formats used compared with the level of customer specific adaptations made. Looking at the number of organizations with a high level of product customization, we see that most of them are found to have relatively qualified product description formats. However, even though the organizations with high levels of customization have more elaborate forms of documentation, they still perceive a lack of documentation of product structure. Conversely, the organizations with no documentation do not perceive much lack. It could be the case, that the structures that are so complex that they warrant use of an elaborate documentation form, are also so complex that they are not caught in the documentation.

The lack of product structure documentation is mainly manifested through increased risk of performing duplicate work, difficulties handling configuration
4.2. Survey on industrial state and needs

**Figure 4.7:** Documentation of product structure used by respondents.

**Table 4.3:** Amount of product customization relative to the types of product descriptions used in the organization, numbers indicating the number of respondents.

<table>
<thead>
<tr>
<th>Product customization (%)</th>
<th>None</th>
<th>Natural language</th>
<th>Configuration descriptions</th>
<th>Informal model</th>
<th>Architecture/component descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-20</td>
<td>2</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>21-40</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>41-60</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>61-80</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>81-100</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>
The Need for High Quality Feature Models

and version management, and communication problems within the organization. Figure 4.8 shows the perceived problems that the respondents see as a result of lacking documentation of product structures. The perceived potential benefits of improving the product structure follow suit in helping the organizations to keep track of and communicate the efforts spent on different parts of the products.

4.2.6 Problem space variability

It is a commonly recounted argument ever since Neighbors’ quote (p. 20), that in order to achieve high levels of effective reuse in software development, reuse has to involve a higher level than only source code. It is also often argued that much time and activity are spent in the early phases of software development, such as requirements engineering, in order to correctly describe the problem space. Many surveys on requirements engineering in industry, e.g. [47, 60], find that larger organizations spend more time on requirements engineering than smaller organizations do. Their reason could be that there is more domain knowledge in a small organization, which simplifies the requirements elicitation, but could also indicate that small organizations have potential to improve the ability to learn from previous problem descriptions.

As seen in Figure 4.9, a majority of the respondents use exclusive problem descriptions for each product. Many of the respondents claim that they face essentially similar problems and requirements for different products (Figure 4.10). Even though many respondents’ products could be suitable for product lines, because of large number of variants based on a common foundation (Figure 4.11), reuse in the problem space is relatively undeveloped in that exclusive problem descriptions are often used.

When asked for indications of shortcomings, as a result of lacking problem structure (Figure 4.12), the respondents are mainly concerned with communication issues resulting in duplicated work. Issues that are particularly interesting for variability management and product lines include, for instance, missed possibilities for reuse, structural problems with implementations, and traceability between requirements and properties. These issues are not rated as high with regard to problem space structure, as they are with regard to solution space structure.

Perceived opportunities for improvements, if there were to be changes to the problem structure, are mainly simplified maintenance, reductions in mistakes and bugs, and increased productivity for developers. As seen in Figure 4.13, the three potentials that rank lowest are improved communication between developers, higher probability for reuse, and higher fulfillment rate of requirements.
4.2. Survey on industrial state and needs

Figure 4.8: Problems due to lack of product structure documentation.

Figure 4.9: Principles used for problem descriptions.
4. The Need for High Quality Feature Models

Figure 4.10: Ratio of problems that are essentially similar between different products.

Figure 4.11: Ratio of products with a common base and individual adaptations.
4.2. Survey on industrial state and needs

Poor traceability of requirements and properties
Structural problems in implementations
Risk of duplicated work
Less reuse
Missed improvement opportunities
Version management
External communication
Internal communication

Figure 4.12: Perceived shortcomings due to lacking problem structures.

Those respondents that see potential for increased reuse are organizations with high levels of customization, but few variants.

4.2.7 Solution space variability

Managing variability in the solution space has essentially been addressed since the first software libraries and similar techniques were developed. More recently, configurable software artifacts, software architectures, object frameworks and many other techniques have appeared. There are many technical solutions available for software developers to use for implementing the variability of their solutions, but less effort has been placed on structuring and modeling of the solutions.

Figure 4.14 shows what proportion of the used solutions in different products that is essentially similar, and indicates the current levels of potentially reusable artifacts in the surveyed organizations. Although most of the organizations are found on the lower end of the scale, there is a notable number of organizations that have a rather high proportion of assets that are prospects for reuse. Figure 4.15 shows the reuse mechanisms used for the organizations’ solution spaces. The most common approaches are variants of clone-and-own behavior, where
4. The Need for High Quality Feature Models

the organizations end up with several versions of their software being concurrently developed. This is consistent with the behavior reported by other surveys such as [45].

Regardless of which structure that is used for the solution space, a majority of the respondents find their current structure to be functional and working satisfactory. Half of the respondents position themselves in the satisfied end of the spectrum when asked to rank their satisfaction with current practice. Only two are distinctly unpleased. (See Figure 4.16.) When asked about problems as a result of insufficient solution space structure, there is no issue that is notably singling itself out from the others. Potential improvements to the solution structure are generally seen as quality improving, in the sense that they have the potential to simplify maintenance, reduce bugs and mistakes, improve testing and validation, but also increase productivity for developers (Figure 4.17).

4.2.8 Perceived needs for changes and improvements

Figure 4.18 summarizes the respondents’ answers concerning what they perceive they need in order to make changes to their current practices regarding the structures of their solution space. The answers for making changes to product
4.2. Survey on industrial state and needs

Figure 4.14: Ratio of solution similarity among different products.

Figure 4.15: Solution reuse mechanisms used by the respondents.
4. The Need for High Quality Feature Models

**Figure 4.16:** Satisfaction level of structures for products, problems and solutions.

**Figure 4.17:** Potential improvements with a well-designed solution space structure.
and problem space structures are comparable to that concerning solution space. Tool support is considered least important for all three aspects, and rather unsurprisingly time and money come out on top for all aspects. Increasing awareness of alternatives and problematization is an option that was chosen relatively often as well. Most needs for methodology and awareness concerned product and problem space rather than solution space, which is in line with the respondents’ satisfaction levels with current practices.

When it comes to the perceived needs of the respondents in order to influence the structure of the problems and solutions that they face and develop, awareness and problematization rank relatively high. Most of the respondents claim that they have some significant influence over the structure, which means that there is potential for improvements, provided that there are applicable and time/cost-efficient approaches available.
4. The Need for High Quality Feature Models

4.2.9 Contrasts with previous findings

There is a substantial number of papers and reports presenting the results of surveys regarding various aspects of software development. They span quite a wide range, both in time and what topics they cover. Aside from surveys, there is also an abundance of experience reports and case studies of various kinds. Most of this material describes practices and events in large organizations. There is however little work that covers the particular aspects and contexts that we discuss in this section, namely variability management for SMEs. We will here present some of the available publications on surveys in reuse and variability, and case studies on variability and product lines.

Laporte et al. [68] report on ongoing efforts to improve the uptake of software engineering standards in very small enterprises (VSEs) having less than 26 employees. They characterize small organizations as economically vulnerable and lacking resources and budgets for software reuse processes. They also state that VSEs focus on developing custom systems, without strong links to other projects, and having a lack of knowledge (or acceptance) of software process assessment and improvement. Rech et al. [97] find that SMEs have weak reuse workflows, and that getting an overview of reusable information is slow and demotivating, due to inconsistent structures, incomplete descriptions, and outdated information. Recording and locating information was problematic, and Rech et al. suggest that wikis could be a possible solution.

Mohagheghi and Conradi [80] provide a review of industrial studies on effects of reuse, with regard to quality, productivity and economic benefits. They cover eleven papers, written between 1994 and 2005, of observational type and not including surveys. The results of the reviewed papers indicate that there are benefits in the form of lower problem density, less correction efforts, and increased productivity to be gained from reuse. The papers also mention that adoption of reuse practices were slower than expected.

Birk et al. [8] contributed a paper presenting the state of practice in product line engineering to a special issue of IEEE Software. The presentation describes five companies of which two are SMEs. The problems and practices described mainly concerned the issues of large companies, and the coordination problems they experience. Concerning variability management, the paper states that three out of five companies perform an explicit modeling of product line requirements. It also states that three of the participating organizations do not see it necessary to maintain links between requirements on core assets and product instances.

In the same journal issue, Graaf et al. [45] describe the practice in software engineering for embedded systems. Embedded systems often have product series and different versions of the products. Graaf et al. found that new projects often
reused requirements specifications from old projects, even for developing a new product line. However, reuse of code and implementations was conducted ad hoc and opportunistically by copying old artifacts.

Few surveys contain information that can be used to get a picture of the variability management, and they generally do not make any distinction between large and small organizations. However, Slyngstad et al. [107] found that half of the 16 developers in an organization surveyed concerning their reuse practices, reused components as-is, while five used components with customizations. A majority of the developers were unhappy with the available documentation of the components and how they interfaced. The conclusions of the work confirm the established opinions about reuse, in that reuse is perceived as improving quality, and saving time and money.

Nikula et al. [86] conducted a series of interviews with 15 respondents from 12 SMEs in Finland regarding requirements engineering practices. They found that the number of supported product variants, not taking into account porting between operating systems, made by the companies was 0 in seven cases, 2 in two cases, and single companies had 10 and 11 versions.

There have been plenty of reported case studies on software product lines, domain modeling, and variability management. More are contributed every year, but only few of them regard small scale development and small enterprises. The reports of successful introduction of software product lines in organizations account for different ways of handling the variability management, from top-down approaches using classic domain engineering, to bottom-up approaches where the current implementations and solution space governs the setup of reusable assets.

A German project involving software product lines in six SMEs has reported on the experiences in [59, 123]. They found that the organizations had potential to gain benefits from using a product line approach, but that documentation and current practices in the SMEs were unsuitable for product line approaches. The organizations’ tendency of making changes for individual customers meant that planning the variability instead of implementing it on-demand was a significant change of viewpoint.

Ahmed and Capretz [2, 1] have looked into the architectural and business practices of larger organizations with an estimated 2000+ employees. Among their findings are that domain engineering, requirements modeling and management of commonality, variability, and architecture artifacts are positively associated with a software product line’s performance. Requirements modeling further increases the understanding of commonality and variability management in products. They also find that strategic planning and market orientation are
4. The Need for High Quality Feature Models

important business aspects that affect the product line’s performance and success.

In summary, interesting and comparable findings of previous work are:

1. SMEs have inconsistent and incomplete structures and descriptions of reusable assets,
2. the perceived benefits of reuse are increased quality and higher productivity,
3. embedded systems developers tend to reuse requirements and copy solutions, and
4. the number of variants is usually low.

We compare those findings to our own below.

SMEs have inconsistent, incomplete structures and descriptions of reusable assets

Figure 4.16 shows the respondents’ satisfaction level with regard to the structures currently used for products, problem space, and solution space. With the exception of product structure, there are very few respondents found on the unsatisfied end of the scale. We see that particularly the solution space is seen as very adequate by most of the respondents. This result is not the same as that of Rech et al. [97].

The perceived benefits of reuse are increased quality and higher productivity

Figures 4.13 and 4.17 show the respondents’ views on what potential benefits that could come from improved structures for, and documentation of, problem- and solution space. We see that the benefits seen by most concern increased quality and higher productivity along with reduced efforts for maintenance. While we did not specifically pose the questions about reuse, the answers reflect the same opinion as that found in previous work.

Embedded systems developers tend to reuse requirements and copy solutions

Graaf et al. [45] based their state of the practice on eight European surveyed companies. In our survey, seven respondents work with embedded software, and four of them claim that requirements are rarely reused. Three of them claim that
they do not reuse any previous solutions. A possible explanation for the difference between our results and those of Graaf et al. is that the organizations [45] are quite large. The organizations that we covered are small and three of them are consultants. An SME that works on consultancy basis in embedded systems could be obligated to hand over the results of their work to the customer, and thus would not have assets for reuse in the same way as larger organizations with products of their own.

The number of variants is usually low

Out of the respondents that answered the question on how many variants of software-intensive products they had, half of the organizations had 5 or less, which corresponds to the findings of previous work. However, we also see that as the size of the organization increases, the number of variants also increases.

Comparison with large organizations

There are thus differences found with regard to the satisfaction of current practices for structuring the problems and the solutions. There appears to be a gap between the two that variability management is intended to fill.

In addition to the SMEs that replied to the survey, we also have replies from four large organizations. They are really too few to discern any trends or tendencies from their answers, but there are many structural differences between SMEs and large organizations, such as staff turn-over, the significance of individual performances, technology sensitivity and longevity, etc. In our results we cannot account for such factors. As previously stated, surveys lack the depth of information given in, for instance, interviews or case studies.

If we make a temporary sidestep from the definitions of the EU and compare organizations with less than 51 employees with those of 51 or more, we can, at least out of curiosity, compare some key aspects and issues. It is reasonable to assume that an organization with over 50 employees would have structural similarities with larger organizations. Placing medium-sized and large organizations in the same group we end up with 22 small and 7 medium/large organizations.

Figure 4.19 shows a comparison between respondents working in small and medium/large organizations regarding some key aspects described in this section. We can note that small organizations, compared to medium and large organizations:

- are active in fewer business domains,
- have less adaptations of the products to individual customers, and
4. The Need for High Quality Feature Models

- to a greater extent construct their solutions for reuse.

Of these comparisons, we particularly note that the smallest companies seem to be good at designing their solutions for reuse. However, the small companies have less traceability between the problem space and the solutions compared to medium-sized and large organizations. This would normally indicate that the utilization of the reusable solutions is less than optimal, but the matter needs more detailed looking into.

4.2.10 Summary

To summarize the current practices in SMEs, with regard to how they handle variability, one can say that while most of the surveyed organizations seem to have an awareness of the benefits of a structured variability management, the level of formalism of the structures used is generally rather low. The means of

![Figure 4.19: Comparative means between small and medium/large organizations. Apart from the value on number of domains, the means are on a Likert-scale where higher indicate more satisfied.](image)
documenting the variability in the organizations ranges from sophisticated to ad-hoc approaches. The amount of similarity in problem and solution space is independent of product variation mechanism.

It appears that the connection of the product structure to the problem space and solution space is generally not reflected upon by SMEs, probably due to the low number of variants that an SME starts out with. When the organization grows, and the variants increase, there is more awareness. The respondents do not generally see structure and management of the product- and problem spaces as significant. Even though the problem space similarity is high, there is little reuse and traceability activities performed.

Although many of the respondents claim that they develop their solutions for reuse, the strongest reuse mechanism is clone-and-own (to the extent that it can be called reuse). We can see a tendency that organizations with either high degrees of customer specific customization or a significant similarity between the products and offerings developed have more capable structures and practices for reuse. However, we cannot see a break-point in complexity of the products, problems and solutions, where the awareness for efficient mechanisms becomes higher.

The respondents are generally more satisfied with the structure of the solution space than with the product and problem space. It is a reasonable result, since one often finds that problems during software development stem from the requirements activities and initial phases where product and problem structures are important.

The larger an organization is, the more it sees problems with communication due to not having suitable structures. There is a tendency in all the respondents to see a proper variability structure as a way to increase productivity and reduce costs based in wasted efforts doing the same thing over and over again. Issues in the problem structure and the product structure co-variate more with each other than with the solution structure.

Based on the respondents’ views that have emerged from this survey, we can identify some interesting opportunities for further research such as the importance of the business domain, compared to the size of the organization. Since there are SMEs that report using variability mechanisms successfully, there are obviously best practices that should be elicited and looked at. While the smallest organizations have few variants, and perceive little need for variability management, the need appears in the growing and expanding organizations. Introduction of suitable variability management mechanisms during the growth phase of an organization could prove very beneficial for practitioners.

It is also interesting to investigate the importance and influence of domain on the practices. Embedded systems, with stronger technical focus, might be
4. The Need for High Quality Feature Models

different from user-oriented systems and software, but we cannot yet tell if it is a strong factor or if it is the size of the organization that is more significant. Finding how the practices change with domain and organization size is a very interesting research topic.

4.3 SEMCO

This section describes the first case study that was part of a research project named SEMCO, where a feature model of a hardware and software family was developed. The feature model described here was developed for a leading supplier of automotive safety equipment. It is an initial feature model, describing the structure of the safety equipment manufactured by the supplier from a customer/end-user perspective. It was developed using documentation from a number of use cases, and extracting the commonality and variability in those cases.

The documentation consisted of the requirements specifications and definitions from a number of projects carried out by the supplier. Each use case provided a customer specification, detailing the needs and demands posed by the customer on the product line.

For each use case there is also a system and software specification of the requirements, describing the actual implementation fulfilling the customer needs. Thus, both the problem domain and the solution domain was covered. The model was developed using the tool Pure::Variants [96].

4.3.1 Context

The application context of SEMCO is the development of software-intensive systems in the automotive industry. Manufacturers and suppliers in the automotive industry of today are facing an increasing set of customer requirements, an increasing need of integration of different electronic components to build complex functionality, and increasing safety requirements. This leads to a large variety of software components for dependable systems in automotives. Figure 4.20 illustrates the increase of sub-systems in cars over the past decades.

A supplier of components to automotive makers is expected to provide different versions of the components adapted to different configurations of the cars. The German car maker BMW indicated in 2004 that only 2 out of 2 million cars produced at its plants have exactly the same configuration of sub-systems, special equipment, or optional parts. Both from a supplier’s and a manufacturer’s

4.3. SEMCO

Figure 4.20: The growth of electric/electronic sub-systems in cars. [120]

point of view, it is of high importance to limit the variability of single sub-systems in order to achieve a reasonable efficiency in development and manufacturing processes.

The purpose of the main research project was to develop methodologies and concepts that applied semantic technologies in model-based software engineering. As part of the solution framework, feature modeling was used as an approach to structure variants in the requirements space.

4.3.2 Objective

This case study was conducted before the core concepts of feature model quality described in this thesis were developed. The case thus acted not as a validation, but as a study into the practical problems of developing feature models. It provided motivation for the identification of problem areas and the needs of industrial practices in variability modeling (thereby its position in Figure 3.1, p. 58).

The development of the feature model for the automotive supplier started with the definition of scenarios for the use of feature models. These scenarios served as a basis in the discussions with the supplier on what feature attributes to include in a feature model. The scenarios were based on already established notions of how feature models can be applied during requirements engineering.
4.3.3 Brief results and value

The feature model is depicted in Figure 4.21 as a graph. In Pure::Variants’ notation mandatory features are decorated with an exclamation mark (!), optional features are shown with a question mark (?), and feature groups are shown with a double arrow.

The model contains relatively few feature groups and restrictions while having a significant number of optional features, which is consistent with the results reported from the development of other industrial feature-models for software intensive systems.

Using the information in the source materials, the feature model was developed for a perspective of the customer. That is, the features seen in the model
are features that are important to the customer stakeholders of the model. The features focuses on the sensors and actuators in a safety system for automotives, and have a granularity that makes it suitable for configuring the different options of the products that are needed by the car makers as they produce various series and configurations of their products.

As Pure::Variants offers the facilities for developing family models of the solution space for the feature models, two such models were also developed. The first family (solution) model, as seen in on the left in Figure 4.22, is focused towards the hardware aspects of the solution space, while the second family model, seen on the right of Figure 4.22, describes a software (runtime) perspective. The runtime system does not describe the current software system for the product line at the automotive supplier, but rather an envisioned modularity, which would encourage reuse.

The family models are developed for the perspective of developers and configuration management staff. The family models are thus not intended to portray end-user visible aspects of the system and product line.

The hardware family model was developed with a component perspective in mind, and focuses mainly on the sensors and actuators found in the system. The runtime family model focuses on the algorithms in the system that transform data between sensors and actuators, such as gyros or accelerometers, and firing squibs of airbags and seat belt tensioners.

In between the feature model and the family models, there are relations that are used for configuring the system in various ways. The relations between the features and the family model principally follow the same types of relations as those mentioned previously, such as inclusion and exclusion. By selecting a feature combination that is validated using the relations in the feature model, parts and components from the family models are included in the system. Depending on how the tools are set up, one can create a build environment, specification, documentation, or configured software system from the feature selection.

The models constructed are just a subset of what would be required for a complete set, capable of supporting a domain engineering oriented workflow. One major part of the workflow that was used in this project was to connect the requirements and solution space to actual artifacts, i.e. creating the relations necessary between the feature model and the family models. This work represent the largest share of the modeling efforts needed to introduce domain engineering in an organization, and requires a lot of domain expertise.

The primary value of the work done in the SEMCO project, applicable to the topics of this thesis, was insights into the wide array of different uses that feature models can have in an organization, depending on the stakeholders. The stakeholders involved in the definition and development of the feature model
Figure 4.22: The SEMCO family models of the hardware and runtime systems.
4.4 Satisfying the need

Based on the presented practices, the review of literature on existing approaches, and the industrial state of variability practices, we can compose a list of high level requirements on an approach for increasing the quality of feature models.

R1 *The method should provide an empirically grounded model for how qualities in feature models are related to one another.*

Past experience with quality models, trade-offs, etc., has shown that quality is often a negotiation issue, where one quality can take precedence over another. It is interesting to explore the dependencies and influences that exist between different traits of a feature model. It has also been found that an issue with approaches to quality in models has been the lack of empirical evaluation of the approaches. Furthermore, in order to be of value to practitioners, the approach should be expressed in terms of a given context.

R2 *The approach should consider the stakeholders’ intended use for the feature model.*

The uses for feature model have expanded beyond the original applications in the telecommunications sector. These different uses for expressing variability pose different requirements on the model and its distinguishing properties. We have found in our research together with practitioners that the expectations of the intended users of a feature model can be disparate, and that feature models offer an opportunity to model and formalize something that in many organizations has been previously intangible, i.e. variability.

R3 *The approach should provide the modelers with guidelines for constructing feature models that are suited for their intended purpose.*

Taking the empirically founded indicators and the consideration of intended feature model use into account, it should be possible to formulate guidelines, recommendations, and best practices for acquiring those properties of a feature model that make a certain quality emerge. An example of existing guidelines for modeling tasks is those for information models [104], that provide accessible advice and recommended practices for creating models that have efficient, correct, and comprehensive design. By following design
recommendations and execute syntactic rules, the modeling process result in better, more suitable, models.

R4 *The approach should provide guidelines and a set of indicators for evaluating the prominence of the qualities in the model.*

In order to have a complete chain of model quality management in place, it must be possible to evaluate existing and newly produced models. This allows the producers of the model to determine whether there is evidence to support that the model possess the desirable quality artifacts, and if it is suitable for its intended purpose. Evaluation and analysis of feature models is also a confidence and theory enhancing task, since it finds, elaborates, supports, or contradicts indicators of prominent qualities in feature models.

R5 *The approach should be possible to integrate with existing methodologies for feature model development.*

Significant work has been made in devising feature models and feature modeling methodologies (as evident from Chapter 2). Many of these methodologies are very capable and are actively pursued and used in various organizations. An approach intended to support increases in quality of produced feature models must be possible to be used in league with the existing development methodologies. This means that the approach to quality assurance should be, within reason, independent of the specific methodology and particulars of how the feature model is produced.

R6 *The approach should be palatable for small organizations.*

Although feature modeling – and software product lines in general – have mostly been employed in larger organizations and for systems of substantial size\(^3\), we have seen that SMEs that are expanding and extending their product offerings find that variability management can be of significant benefit. There is a tendency towards products needing to be highly customized and adapted to different user segments. This is pushing the software elements of products to provide the difference in the user experience. Small organizations are often increasing the complexity of the software without variability management until it becomes difficult to handle. Having the ability to introduce variability management into their practices at an earlier stage could remedy some of the emerging problems.

\(^3\)See for instance the Software Product Lines Conference’s Hall of Fame: http://splc.net/fame.html
4.5 Chapter summary

R7 The approach should take into regard the domain of the model and the requirements posed on the model because of the domain.

There is not enough empirical material or related work to indicate clearly whether or not the technological domain is an important distinction for how different organizations handle variability. It is however a fact that the domain influences the needs and practices for many other parts of the development process, such as requirements engineering, implementations, testing, documentation, etc. It is thus not unreasonable to consider the demands that a particular domain could pose on the qualities that emerge in the variability of the product-, problem-, and solution spaces.

The remainder of this thesis details a proposal for construction, evaluation and planning for achieving high quality feature models, its application in industrial use cases, and its refinement. The above list of attractive traits for such a proposal – although not exhaustive – is considered in the designs of the approach.

4.5 Chapter summary

This chapter has covered the current state of quality consideration in existing feature modeling practices, and the perceived need and potential of feature modeling, as seen by SMEs. When evaluating a representative selection of feature modeling practices, several criteria for model quality are found to be absent from several of the methods. Reuser involvement, scenario consideration, feature model validation and verification, quality attributes considered, and metamodels and notations are showed to have varying, but essentially insufficient support in existing practices. There is room for improvements, and there are approaches in other modeling areas that can be used for inspiration.

A survey of software developing organizations shows that the variability management practices in SMEs are quite disparate. Modeling of variability is still relatively uncommon, but many are facing the problem of variability management and can see benefits from improved practices. Most problems of variability are seen on the requirements side, as opposed to the implementation side. A case study in the domain of embedded software for automotives, suggested that there are several applications for feature models that can depend on the stakeholders and the envisioned usage scenarios for feature models. Based on the findings from the case study, the survey and the methodology review, the chapter concludes with a set of high level requirements for an approach to raise the quality in feature models.
4. The Need for High Quality Feature Models
Initial Quality Model and its Application

To ensure that proper attention and effort is given to the crucial parts of the feature model under development, given a stated goal and purpose of the model, the effects of properties of the feature model on different qualities need to be explored. For example, having a model with high accuracy and correspondence to the product family can be expected to have a positive influence on usability, and make the model easier to communicate and learn. However, a model with plenty of details and information could also be harder to modify and evolve. Depending on how the feature model will be used, decisions on what quality attributes that have priority should be made.

This chapter describes the first iteration of a model for feature model quality, the application of the model in practice, and the empirical basis for the quality indicators. This work resulted in a second iteration of the quality model and the methods for using it. The first iteration was based on the feasibility study of already existing quality models. A transformation and adaptation of the existing quality models were made, and a quality model that was suitable for feature models was formulated.

The ideas of this iteration were then used and applied in the case study at Tactel. The case study resulted in a set of indicators that can be used to

find the prominence of a certain quality in a feature model. It also refined the procedure for performing the quality targeting, the evaluation of the resulting feature model, and gave rise to a feedback loop of quality indicators refining the quality model.

The second, refined, iteration of the quality model and the associated methods and procedures are presented in the next chapter.

5.1 First iteration of the quality model

It is possible to assemble a list of some hundreds of attributes, metrics, and properties suggested in the other quality models that have been published (section 2.3). While such a list can serve as inspiration and a stepping stone, it is of course unpractical and, in large parts, not applicable. The first step in developing a quality model for feature models is thus selecting a reasonable set of factors and attributes to work with, based on the past efforts. However, since most quality models are not constructed to deal with models, we also need to redefine the meaning and interpretation of some quality attributes to suit the nature of models. Although the names and terms would appear similar to other quality models, some meanings change significantly when applied to feature models.

Establishing and evaluating relevant properties and characteristics to be investigated, is followed by determining how to establish metrics and procedures for performing the measurements on feature models. Past work in software metrics has focused extensively on sizes and numbers of implementation artifacts. Most of today’s widely recognized metrics are thus not necessarily appropriate for models, and we again need to make definitions and evaluations aimed at models. In the remainder of this section we describe the selection and organization of qualities and attributes for the initial quality model. This iteration of the quality model was published in [115].

The aim of the proposed model is not to get a single, normalized value for the quality of the feature model as a whole in order to compare it to any other model. Instead we aim to find indications on what qualities the feature model possess, at the expense of other qualities. As mentioned and motivated in previous chapters, an important aspect of our approach to feature model quality is the notion of intended purpose of the feature model. Thus, a single number for comparison of two feature models would only be of interest if the intended purposes are the same, and the priority of the required and requested qualities is the same. The suggested approach for comparison of feature model qualities is explained in section 5.2.11.
5.1. First iteration of the quality model

5.1.1 Overview

In order to apply the quality factor modeling concepts to feature models, we need to establish a few other concepts and terms in order to avoid confusion (recall the list of issues in section 3.2). Figure 5.1 shows, informally, how the elements of the initial proposed quality model relate to one another. There are eight principal components of the proposed modus operandi.

Quality factor Six top-level quality factors contribute to the overall quality of the feature model. The quality factors embody the main properties that can be prominent in a feature model. Given a context in which the model is intended to be used, the developers of the feature model can in the six main qualities identify which priority should be given to different aspects of the development of the feature model.

Perspective The quality factors are seen from the point of view of the developers and users of a feature model. The selection and formulation of quality factors in the initial model is based on this viewpoint. Given a different perspective, different quality factors could be more relevant. It is important to understand the distinction between regarding the quality of the feature model itself, and the resulting products that are modeled.
5. Initial Quality Model and its Application

**Quality attribute** As in many other quality models, both regarding models as well as other software- or physical artifacts, each quality factor has a set of specializations called quality attributes. The attributes are refinements of the main factors that can be operationalized and evaluated in order to find the contributions to the quality factors. An attribute can provide a positive or negative contribution to one or more of the quality factors, but for convenience they have a main association to one of the six factors.

**Indicator** Actual measurement or evaluation of a quality attribute is performed on indicators. Each quality attribute can have several indicators that influence its relative rank. The indicators are the operationalized values of the quality in a feature model and thus express the quality attributes. In the same way that a quality attribute can influence several quality factors, an indicator can influence several attributes.

**Method** The capture of a quality attribute can be done in several different ways. Some of the attributes lend themselves well to be determined by static analysis of a feature model’s structural layout, essentially being functional attributes, captured by functional and objective indicators. Other attributes are more subjective, and require an evaluation process where an opinion or assessment is made by users or modelers. The different types of attributes require different methods to capture them, and the nature of the method used influences what kind of indicators that are available.

**Measurement action** The indicators have to be captured with some form of measurement action, suitable for the different attribute types. Each measurement uses a procedure to arrive at some value for an indicator and has instrumentation, templates, guidelines, or other recommended practices for producing the result of the measurement action.

**Context** The context of the modeling efforts influence the capture of indicators. It could be the case that an indicator has different measurement actions associated to itself. The context of the feature model and the evaluation could determine or influence the selection or parameterization of methods or measurements. Another matter to address, that is context dependent, is who should perform the measurements and how to integrate the results in the modeling process.

**Result** The result of the measurement action is a value that shows in absolute or relative terms how prominent an indicator is in the feature model. Depending on the method used and the measurement actions that are performed, the result can be on different scales. Structural properties in a
5.1. First iteration of the quality model

Figure 5.2: Intuitive mapping of model elements in ISO 9126, the initial proposal for feature model quality, and the quality model by Boehm.

A feature model would likely be of absolute or relative scales, while subjective indicators could result in nominal or ordinal scales.

Figure 5.2 shows an intuitive comparison of the terms introduced for the initial proposal and the terms used by ISO 9126 [54] and Boehm [9] in their quality models. In order to illustrate the level of abstractions that each term encompass, some examples for each term have been added to the figure. Obviously, the terminology in the initial model would have been closer to that of previous quality models if attribute in the proposal had been called characteristic. The reason for the initial terminology is that it avoids the unclear distinction between primitive and metric in the ISO 9126, but having each element of the model hold a descriptive name. The positioning of the proposal’s terms between the two other models in Figure 5.2 is thus intentional, since it combines the more meaningful terms from both models.
5. Initial Quality Model and its Application

5.1.2 Qualities and quality attributes

The initial proposed model for feature model quality contains six top-level quality factors, with 25 attributes that contribute effects to the quality factors.

Table 5.1 shows the top-level quality factors and a sorting of the attributes under each factor. The sorting is based on where each attribute provides its main contribution, but should not be seen as an immutable distribution.

The remainder of this subsection provides descriptions of the quality factors and the attributes. The descriptions here have been edited for clarity compared to those found in [115].

**Changeability**  The quality factor describing the model’s ability to evolve as the modeled product line evolves, while maintaining the same purposes and applications as previous iterations of the model. A model describing a product line that is expected to change rapidly or regularly must have the flexibility to accommodate those changes, without requiring too extensive efforts for re-design.

**Adaptability**  This the ability of accommodating changes to the existing model, in the sense that existing constructs are modified. Adaptability is an important trait for a model to allow reconstruction without losing structure and recognizability.

**Extensibility**  This attribute describes the ability to accommodate additional constructs to the model. The ability to add features to a model without significantly changing the abstraction level, stakeholder interests and intention of the original model.

**Stability**  This attribute denotes the perceived change expectation. A stable model is developed with the intention of covering the foreseeable evolution of the modeled product line with relatively little changes to the model. A model developed with this focus is less likely to gracefully handle major changes.

**Reusability**  This quality factor contains the properties necessary to enable or facilitate the reuse of the current model when evolving or developing other models. The level of reuse can be on different levels of granularity, such as entire models or partitions of it.

**Modularity**  The attribute that determines to what degree partitions of the feature model can be extracted and repositioned. The model’s ability to be divided into meaningful composable parts is the essence of this attribute.
Table 5.1: Quality factors and attributes

<table>
<thead>
<tr>
<th>Factors</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Changeability</td>
<td>Adaptability</td>
</tr>
<tr>
<td></td>
<td>Extensibility</td>
</tr>
<tr>
<td></td>
<td>Stability</td>
</tr>
<tr>
<td>Reusability</td>
<td>Modularity</td>
</tr>
<tr>
<td></td>
<td>Self-containedness</td>
</tr>
<tr>
<td>Formalness</td>
<td>Analyzability</td>
</tr>
<tr>
<td></td>
<td>Conformance</td>
</tr>
<tr>
<td></td>
<td>Consistency</td>
</tr>
<tr>
<td></td>
<td>Testability</td>
</tr>
<tr>
<td>Veracity</td>
<td>Accuracy</td>
</tr>
<tr>
<td></td>
<td>Redundancy</td>
</tr>
<tr>
<td></td>
<td>Completeness</td>
</tr>
<tr>
<td></td>
<td>Reliability</td>
</tr>
<tr>
<td></td>
<td>Robustness</td>
</tr>
<tr>
<td>Mobility</td>
<td>Installability</td>
</tr>
<tr>
<td></td>
<td>Interoperability</td>
</tr>
<tr>
<td></td>
<td>Portability</td>
</tr>
<tr>
<td>Usability</td>
<td>Complexity</td>
</tr>
<tr>
<td></td>
<td>Learnability</td>
</tr>
<tr>
<td></td>
<td>Structuredness</td>
</tr>
<tr>
<td></td>
<td>Understandability</td>
</tr>
<tr>
<td></td>
<td>Acceptability</td>
</tr>
<tr>
<td></td>
<td>Accessibility</td>
</tr>
<tr>
<td></td>
<td>Communicativeness</td>
</tr>
<tr>
<td></td>
<td>Visibility</td>
</tr>
</tbody>
</table>

*Self-containedness* This attribute describes the model’s ability to remain unaffected by changes in other models, and to what degree the model is dependent on external models. This attribute is only valid if the model exists in an environment where there are models that have links to one another.
5. Initial Quality Model and its Application

**Formalness** This quality factor contains the properties related to the ability to manage the model in a formalized manner, notably for the purpose of machine management, tool support, and automation.

**Analyzability** This is the degree to which rules of composition are formalized in the sense that tools can reason and determine valid configurations. High analyzability indicates that there is less need for domain experts to decide on reasonable or correct configurations. This attribute is dependent on the possibility to express the real possibilities for composition in a formal language.

**Conformance** This attribute denotes the degree of conformance and agreement with standards, meta-models, and formalized modeling rules. This attribute determines several important aspects of a feature model with regard to not only formalness, but also usability aspects such as understandability and evolution.

**Consistency** The attribute that determines the absence of contradictions in the model. A model could contain legal but contradicting constructs, leading to less analyzability. This attribute also influences other quality factors such as usability.

**Testability** This attribute denotes the extent to which the feature model can be tested for validity, using essentially automatic reasoning. A decisive indicator is of course tool support, but also the constructs, formalness and syntax provided by the meta-model for the feature model. Manual approaches to testing the model are not covered by this attribute.

**Veracity** This is the quality factor that relates to how well the model corresponds to the real world, i.e. it is the degree of precision in mapping the modeled product line to the abstractions in the model.

**Accuracy** This attribute describes how well the model represents the actual world. There are obvious effects to this attribute in that a high accuracy is expected to result in a more applicable model. However, high accuracy could lead to less user-friendly models, due to unsuitable level of abstraction or design.

**Redundancy** This is the measurement of how concise the model is, i.e. to what extent the model is the smallest representation of the product line, given the intentions of the model. The reciprocal of this attribute would be the parsimoniousness of the model. Parsimony is
5.1. First iteration of the quality model

often seen as a desirable trait in models, but there could be cases where redundancy could affect usability.

Completeness This attribute denotes the state of the model with regard to how well it covers the product set that constitutes the product line. A product line could contain configurations that would be exceptions or extraordinary configurations, which are not handled gracefully by the mapping between reality and feature model. This would mean that the product line could have variations that are not covered by the feature model.

Reliability This is the level of confidence in that the feature model will result in configurations that will perform their intended function and fulfill their designated purpose. Depending on the context in which the model is used, there could be considerations for how dependable and formal the model needs to be. Communication purposes and generative purposes could pose different requirements on how the model is constructed.

Robustness This is the model’s resistance to resulting in incorrect configurations due to misunderstandings of the model. While a model might contain valid constructs that contradict intuition, this attribute is the aspect of the model containing possibilities for making configurations out of misunderstanding or of incorrect use of the logic, design, or constructs in the model.

Mobility This quality factor denotes the ability of the model with regard to how effectively it can be put into productive and meaningful use.

Installability This attribute determines the model’s ability to be made available to its users. Aspects affecting this include different meta-models and tools without much coherence between different options. Besides purely technical issues, there are practical and economic consequences to consider.

Interoperability This attribute relates to the model’s ability to co-exist and be used in conjunction with other models, for instance the technical interoperability of models and model repositories.

Portability This attribute is the model’s ability to be moved between different environments. It is a special case of interoperability, in the sense that it represents the vulnerability to changes in the supporting environment and tool changes.
Usability  This quality factor collects the properties of the model that affects
the user’s ability to utilize and receive beneficial effects of the model.

Complexity  This is the technical complexity of the feature model, which in
turn affects the other usability attributes. This attribute encompasses
the basic quantitative measures of a model. It is however independ-
ent of the presentation such as layout and tools, notations, etc.

Learnability  This is the property of the model that relates to how much
effort a user of the model must spend in order to learn how to use
the model for meaningful applications. The learnability represents
the effort needed to understand the model, regardless of notation or
presentation.

Structuredness  This is the property of the user being able to detect recogniz-
able patterns or meaningful structures in the model. The structure
of the model could follow a pattern of logic or abstraction in a way
that makes the model recognizable due to pre-existing knowledge of,
and familiarity, with such patterns.

Understandability  This is the property that decides the extent to which a
user of the model can access its meaning and intention, without the
need for excessive domain knowledge, learning efforts or similar, and
the ability of the model to convey its scope and capabilities. Smaller
feature models may communicate its more specific domain scope
more efficiently than larger models that could encompass more do-
mins or use more advanced modeling constructs.

Acceptability  This is an attribute to denote the tendency of the model to be
adopted and accepted within an organization for use in the intended
way and for the intended purposes. The acceptance attribute is in-
fluenced by the intended users’ perception of how well the model
corresponds to their needs, and the effort needed to include it in the
regular work routines.

Accessibility  This is the ability of the model to be used by a certain user
base. Factors of a user’s ability to use the model are based on the
user’s prior experience. A model designed for high accessibility would
aim to reuse as much of the intended user’s background knowledge
as possible.

Communicativeness  This denotes the model’s ability to communicate its con-
tents to its users. This is the attribute related to the presentation of
the model, meaning that a model could be highly communicative,
5.1. First iteration of the quality model

even though it is not necessarily well suited for the intended purposes.

Visibility  This is the model’s ability to reach its potential users. A model with high visibility has a greater range of applicability of the contents, level of detail, and possibility to communicate to stakeholders.

5.1.3 Indicators

In order to have the attributes measured, we use a set of indicators that are associated to the attributes. Each indicator can be significative for several attributes and give a positive or negative contribution to the quality attribute. Each indicator uses a quantitative or qualitative action to capture the indicator, depending on the methods that can be applied to the indicator. Existing quality models and practices for metrics on software and other products (for example [61]) list several different metric types, using different scales of measurement such as numeric values, boolean values, presence or absence of some property, high/moderate/low occurrence of a property, etc.

In the case of feature models we can find correlations between quality attributes and simple indicators, such as the size and structure of a model or what tools, notations, and meta-models that are used. For some qualities we need to investigate perceived user experiences which are not captured through a quantitative metric. For example, among the indicators for Accuracy we could use

i) degree to which the granularity of the model corresponds to the granularity of the actual product line, i.e. abstraction level of features compared to offered products and options,

ii) to what extent composable features in the feature model represent composable artifacts in the solution space,

iii) agreement between terminology and feature model, and use of domain-specific language, and

iv) the preciseness of dependencies or compositional rules with regard to solution space equivalents.

The examples given for accuracy are quantitative, in that they would all result in ratios or percentages. Meanwhile, indicators for Installability can include tool support and representation or repository formats, but also meta-models used and the training level of the end-user of the feature model. These indicators would result in metrics that would describe which choices that have been made among a number of alternatives, thus also quantitative, but on nominal or ordinal scales.
5. Initial Quality Model and its Application

5.1.4 Relationships and trade-offs

Let us take accuracy as an example for how quality attributes affect quality factors. A high accuracy of a feature model is expected to have a positive influence on attributes of the usability quality factor. Accuracy in the depiction of the product line should make the model easier to communicate and learn. On the other hand, high accuracy could affect changeability negatively, as it would make the model stiffer and less amendable for evolution. This is an example of how there are relationships between the quality attributes and quality factors, where the attribute can provide positive or negative contributions to different attributes and factors. This means that there are trade-off decisions to be made for a feature model. Raising the quality on one end will have consequences for others. The process of determining which qualities to put ahead of others is a negotiation activity between the stakeholders of the model, as represented by “perspective” in Figure 5.1.

5.2 Case study: Tactel AB

The initial quality model for feature models was drafted on the basis of existing quality models and prior work on finding quality evaluation methods for models. Although there was some practical foundation in the hints taken from the experiences in the SEMCO use case, the contents of the model was mostly theoretically derived and motivated. This section describes the application of the initial quality model in the Tactel use case. The results of the application were analyzed and interpreted, leading up to the second iteration of the quality model, which is described in chapter 6.

Before describing the process, experiences and results of the use case, we briefly digress on paired comparison, a method that was used in the application case.

5.2.1 Paired comparisons

In this study we use paired comparison for determining a ranking of what the most important and prominent qualities are in a feature model. Paired comparison is a common method for comparing choices and arriving at a preference. It is a method that is easy and fast to administer and provides a straightforward way to gather the collective ranking of a group of developers or other stakeholders.

Nunnally and Bernstein [87] discuss absolute versus comparative responses. They claim that because of a frame-of-reference problem in absolute responses, people are almost invariably more consistent and accurate in making compar-
isons. Part of the reason for this is that there are few situations in daily life when absolute judgements are made.

Given a set of options or items that are to be compared, in our case qualities, a matrix is constructed similar to the example in Figure 5.3a. Each stakeholder in a group fills out the matrix by selecting which item in the intersections that it considers the more important (Figure 5.3b). Finally a summation is made of how many times each option has been selected among the stakeholders and a ranking order, also called preference score, is found (Figure 5.3c). This order is ordinal at minimum, but generally accepted as having interval properties [12].

In our study we first use paired comparison to get a ranking of what qualities that the stakeholders of a feature model regard as the most important. We also use it for evaluation purposes where a stakeholder is presented with a feature model, and is asked to use a comparison matrix to appreciate which quality that prevails in the model. The remainder of this chapter details how the study was conducted.

5.2.2 Case study context

Tactel is a company that develops mobile applications. A mobile application is typically expected to run on a multitude of different handsets (phones), each having its own combination of limits on display size, means of input and other hardware characteristics. Mobile applications are increasingly using external data feeds or other communicating services. Examples are streaming media, RSS-feeds, stock tickers, weather and traffic information, maps, etc. This information is acquired from content providers, often via subscription based services that vary in form and terms between carriers and providers. The operating system and target platform also place requirements on available programming languages, binary compatibility, usable libraries, and user interfaces.

Given the amount of variability in the mobile applications domain, issues of portability, cross-platform development and compatibility need considerable
5. Initial Quality Model and its Application

attention. Software product lines present a promising approach to solve or alleviate some of the issues. Since the cell phone industry is already offering their handsets in terms of product families and product lines, and have also made inroads to use SPLs for their device specific software [75], it makes sense for developers of mobile applications to look to software product lines as well. Makers of user-level applications for mobile phones are often SMEs. Such organizations do not have the same level of elaboration when it comes to development processes and product line approaches [125], but face the same level of variability that is found for the phone manufacturers.

In [3], experiences are reported from a product line of games for mobile phones. The authors find that the mobile game domain (a subset of mobile applications) is characterized by short development cycles, high variability, difficulties porting the games to a huge variety of platforms and handsets, and that the domain has many SMEs. One of their results is a hypothesis that the companies in the mobile games domain have often adopted a software product line approach without knowing it, an observation that we come back to in a while.

5.2.3 The company

The company, Tactel AB [111], involved in the case study is an SME with several geographically dispersed development offices and additional sales offices. Each (development) office is typically a size of 20-25 persons, and most projects are carried out at a single office since each location has special competence in particular languages or applications. As the company grows its product portfolio, it is adapting its development processes and resources to accommodate increasing complexity and variability.

Tactel’s customers are handset vendors, network operators, and content providers. Figure 5.4 shows the business drivers as identified by Tactel. A customer typically outsource an entire project to Tactel, which develops the applications on the handset side, while the customer provides services on the server side. Product management and deployment are then taken over by the customer.

The business domain that Tactel operates in poses high demands on effective and rational development. Given the requirement for short time to market, reuse is imperative in order to alleviate some of the constraints and problems that the organization faces. As a means to do this, the company has developed an internal framework of reusable components, called plugins. These plugins work as elements of an application using a runtime system, which principally works like a browser, interpreting the application written as XML-documents and presents the application to the user (Figure 5.5). Creating an application
5.2. Case study: Tactel AB

1. A mobile handset at the customer...
2. ...and a functional mobile network with subscribing customers...
3. ...and interesting content available from the terminal to the consumers.

Figure 5.4: The Tactel business drivers as illustrated by Tactel. [111]

Figure 5.5: Overview of the communication flows in the framework for mobile application development.

with the framework consists of writing platform independent XML, that runs on top of the executive on any of the supported operating systems and handsets.

5.2.4 Objective

The core plugins support a common set of functionality that is used for most of the developed applications and projects. To achieve functionality (be it log-
ics, presentation, interface, or communication) beyond what the core offers, new plugins are developed from scratch or based on the plugins found in the core. Over time, a large variety of plugins have been developed for various applications and projects. Thus, as hypothesized by [3], we also find that the company has, because of the intrinsic variability it faces, already applied advanced reuse and product line concepts, without explicitly terming it so.

As the number of developed components that are spread across various projects is becoming too large to overlook using trivial methods (simple lists, spreadsheets), the risk for duplicating efforts grows. If the company would lack efficient and applicable methods to locate and reuse components, they would also miss out on the potential increase in quality that one get from using tried and tested components. Feature models meet the demands for a solution.

The research questions that the case study contributed towards answering can be formulated thusly:

Q1 What quality factors should be compared?

Q2 How can paired comparison be used for evaluating the quality in feature models?

Q3 What are the indicators of quality in a feature model?

Q4 Can prioritization be used as a means of directing the modeling activities to reach desired qualities in a feature model?

We address the research questions by applying prioritization and evaluation to the feature model and found, as reported in subsequent chapters, that there are indicators that contribute positively or negatively to qualities in the model, and that the factors are adjustable.

The project was expected to result in methods, tools, processes, or artifacts that would contribute to more efficient management of variability aspects in Tactel. A criterion used for scoping was that as many as possible in the organization should benefit from the work conducted.

5.2.5 Planning and preparation

We refer to a technical report [116] by the author for details on project planning and execution.

Since the project was carried out in cooperation with staff from academia with experience and expertise in feature modeling, the training of the modelers was mainly carried out during workshops, based on various presentation material. Feature elicitation techniques could be kept rather straight-forward, thanks
to the well-defined scope of the feature model. The modeling notation was introduced using generic examples, based on the capability of the tool used for the modeling.

Initial meetings to decide on the scope and size of the feature model were held on a project manager level, gradually including more staff from the organization as the scope was narrowed down, and the necessary expertise was needed. Once the scope was determined and the participating projects were selected, workshops were arranged to gather the project members.

In preparation of the first workshop, the modeling leaders had reviewed the documentation for one of the participant projects. This initial review resulted in each of the existing plugins being assigned a number of single-word tags, describing characteristics, main functionality, or variant functionality of the asset. The tags were then used to seed the model with a basic structuring, as several plugins were identified having similar functionality or being variants of similar functionality.

Undoubtedly, this initial analysis of one project, made by the project leaders, affected the rest of the analysis by the participating projects. Although the model was entirely open for changes in the workshop stage, most of the analysis and structure from this first seeding remained into the rest of the modeling. The question is if this had a positive or negative effect on the overall resulting model. While a positive effect was a running start for the first workshop, it could also have lead the developers into tracks that were inappropriate for the domain and scope.

In the end, as also noted in the next section, the end-users of the model were satisfied with the structure and content of the model, so we would still recommend this practice, provided that it is performed by persons with adequate understanding of both variability modeling and the domain.

Initial analysis and modeling of the assets were made using pen and paper, in order to get the developers thinking in terms of features, and quickly get started on the modeling notations. After the first workshop, a basic structure of the assets had emerged and could be used as a basis for further modeling. The assets originating from the projects were then added to the model, based on the general structure. Once all the plugins from the projects had been added to the model, a series of smaller workshops were held in order to validate and evaluate the resulting model.

5.2.6 Scoping

The model scoping activity aims at finding the boundary of the model and setting the goals for the outcome of the modeling activities. The input used to
decide on scope usually include the organization’s business and product strategy, technical suitability, access to domain expertise and sources, etc. Ideally, the scope of the model should end up encompassing enough to provide the most value for the organization, but still be manageable and prone to be realized. The recommendations found in various guidelines and methodologies are to pick a scope where the included parts are well understood, relatively small and not subject to pressuring schedules, which would mean that the analysis would be rushed.

In the case reported in this paper, the selection of the framework and the various plugins as subject of feature modeling was chosen, since the plugins generally represent components suitable for reuse, and are often made in different variants providing somewhat different functionality. Having a proper and common structure and overview of the plugins would benefit the possibilities to reuse previous efforts, reduce the risk of duplicating functionality in different locations and projects, and benefit a large number of people in the organization. It also had the additional benefit of being easy to formulate in measurable progression, i.e. keeping track of how many plugins and projects that had been included in the feature model.

Practically, the selection of the plugins to be analyzed for commonality and variability, and be included in the model, was based on the documentation level that existed for the plugins. While, obviously, it is desired that all plugins and derivations that are produced are documented, time pressure and available developer resources result in varying levels of elaborate documentation. For the initial round of analysis and modeling, four projects that produced plugins and variants of plugins were chosen, based on the level of documentation that the projects had produced and made available. Three of the projects were managed in the same company office, which also made it easier to gather expertise concerning the components.

5.2.7 Tooling (FMWIKI)

Almost all authors and methodologies in the field of variability management and analysis recognize the problem of scaling modeling and visualization of variability [25]. When the complexity of a product line gives rise to many interdependencies and relations between features, and when the number of features in the model increases, most notations for visualizing models break down and become very hard to use. Keeping track of all the data and the changes that go into the model becomes impractical using trivial methods and approaches, like text documents, schematic drawings, lists, etc. Some form of dedicated tool support
becomes a necessity in order to handle the large amounts of information that go into the models.

However, it is not only the visualization that poses problems to the up-scaling of variability modeling. Many of the examples in texts about feature modeling use constructed examples that are not very large and could probably be managed in a rather simple manner. In practice, the variability models contain many more features and have more information tied to them than what is visible in the overviews of feature diagrams. Management and maintenance of the models become as hard as it is vital, if the tools and methods used are not equipped for continuous modeling and updating.

In order to develop the feature model at Tactel, a tool had to be chosen. While there are a few options for commercial tools that could have been used in the project, it was decided that a different tool was needed. This resulted in FMWIKI, a web-based feature modeling tool. The tool represents an engineering contribution of the work in this thesis, in that it was created based on an analysis of the practical circumstances in Tactel. The requirements on tool support for feature modeling in a distributed organization can be interesting for other practitioners as well.

As previously mentioned, Tactel has development and sales offices in several geographical locations. Development of reusable assets is conducted in different sites and in several projects and the assets are used by different projects, meaning that a model of the assets needs to be constructed and updated by, and communicated to, several locations and staff in many different roles. Aside from the modeling and updating of the feature model by staff in different locations, there is the matter of distributing the model to the end-users. Collecting changes and updates and making them available in a timely, consistent, and convenient manner pose demands that can be hard to meet using desktop applications.

Rech et al. [97] have suggested that wikis could be a suitable basis for reuse-oriented tools. The benefits they found in their application of a purpose-built wiki in development projects, of interest in the context of this report, include the use of a single point for storage and retrieval of information and a low usage threshold. The main drawback found was the possibility of opening up for generally low quality documentation due to incompleteness and inconsistencies.

A purpose-built wiki-style modeling tool would fulfill several of the requirements posed on a tool for feature modeling in Tactel. It would be easily distributed via a web interface and support concurrent modeling and immediate distribution of changes to all users. Thusly, a web-based tool for feature modeling was implemented based on a flexible meta-model and a practical notation decided on during the scoping activity.
5. Initial Quality Model and its Application

The meta-model is implemented in an SQLite database, and the web interface is made in PHP. While the meta-model, and thus the repository database, is very pragmatic and flexible, the web interface is tailored to correctly interpret the information about features, associated attributes, and relations to support the aims of the organizations.

The graphical notation is controlled with CSS-templates and can be changed to match other preferences. FMWIKI supports user administration, model administration, defining feature attribute types and relationship types, browsing the feature tree and relationships, adding, editing and removing elements of the feature model, filtering views and full text searching in the model.

At the time of this writing, the implementation is feature complete and functional, but there are many ideas for improvement.

5.2.8 Prioritization

In order to address research question Q1, an inspection of the initial quality model described in this chapter was made by the modeling leaders to decide what the most important emergent properties of a feature model are.

As previously mentioned, the initial quality model for feature models had six quality factors and 25 associated quality attributes. For the application of the quality model in the case study, one quality factor was renamed from veracity to correctness. The meaning of this quality factor is still the same however, i.e. we mean the semantic correctness with regard to the domain under investigation, rather than the syntactic correctness, which is covered by the formalness quality factor. In this case study, we limited the scope to evaluation on quality factor level rather than on quality attribute level. The following six quality factors were thus used in the prioritization and evaluation steps of the project:

- Changeability
- Correctness
- Formalness
- Mobility
- Reusability
- Usability

Before commencing with the modeling sessions, each stakeholder in the case study filled out a pairwise comparison of the six quality factors. This resulted in a ranking, seen in Table 5.2, of what qualities the stakeholder group saw as the
most important in the eventual feature model. The rank value is calculated as the total number of tokens placed on the quality divided by number of stakeholders, giving the average number of times the stakeholders chose the quality in the pair-wise comparison.

Usability leads by a wide margin, followed by reusability, changeability, and correctness close together. We find formalness last, a result that we will come back to later in the thesis.

5.2.9 Construction

At the initial workshop introducing the feature modeling project, almost all the team members from the different projects were gathered and the modeling initiated. Although the participant projects had adequate documentation available for the modeling purposes, the developers of the assets to be modeled had unique expertise and familiarity with the components, and were thus involved in the analysis of the variability provided by the plugins and their derivations. A basic structure of the assets resulted from the initial modeling workshop and was used as a basis for adding the project-specific assets.

For comparison with other feature models, some numbers describing the general properties of the resulting model can be seen in Table 5.3. The validated model contains 305 features, distributed over 9 top-level features and 62 variation points, and has a maximum depth in the hierarchy of 5 levels. The model is thus quite shallow with many variation points. Intuitively, this corresponds to a structure of a model with fewer common features and more focus on the variability. The model in this case does in fact have few mandatory features, but many optional features and feature groups.

\footnote{For example, the two models described in [32] have 300 to 350 features.}
5. Initial Quality Model and its Application

Table 5.3: Some properties of the resulting model.

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of features</td>
<td>305</td>
</tr>
<tr>
<td>Number of top-level features</td>
<td>9</td>
</tr>
<tr>
<td>Maximum depth of hierarchy</td>
<td>5</td>
</tr>
<tr>
<td>Number of variation points</td>
<td>62</td>
</tr>
<tr>
<td>Number of relations</td>
<td>11</td>
</tr>
<tr>
<td>Number of relation types</td>
<td>3</td>
</tr>
</tbody>
</table>

Once all projects had added assets and variants to the feature model, each project validated the feature model together with the modeling leaders, clarifying modeling choices and loose ends. Each feature was annotated with a description of the main functionality in the feature, the origin of the feature for tracing which project it was developed in, restrictions, and external dependencies.

During the modeling sessions and the validation sessions, several benefits of having a variability model became apparent. Different projects had developed assets with the same functionality, unaware of other plugins with the desired functionality. Several relations and dependencies among the assets, that were not previously explicit, were indicated and documented.

5.2.10 Evaluation

During the validation, several dependencies that had gone unnoticed were detected. Many of the missing relations were found as a result of the modeling leaders noticing inconsistencies or missing elements. Of course, the developers that participated were not experts on feature modeling, nor was the intention that they should devote too much time for training purposes. It is expected that the level of modeling proficiency will rise as the model is put to use and maintained by the organization.

The model was also evaluated by the participants, in order to see if there were any major arguments against the current state of the model. The participants were satisfied with the structure, and found it to adequately describe the current state of the scope, be usable for the purposes intended, and containing appropriate information. However, a proper evaluation of the model will require that it has been used for some time.

Once the modeling sessions had been completed, and the resulting feature model was validated in a series of smaller workshops, the stakeholder group was
Table 5.4: The average number of tokens given each quality factor during Evaluation. Higher number indicates higher presence of quality factor in the feature model.

<table>
<thead>
<tr>
<th>Quality factor</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>Changeability</td>
<td>2.8</td>
</tr>
<tr>
<td>Correctness</td>
<td>3.1</td>
</tr>
<tr>
<td>Formalness</td>
<td>1.6</td>
</tr>
<tr>
<td>Mobility</td>
<td>2.6</td>
</tr>
<tr>
<td>Reusability</td>
<td>1.6</td>
</tr>
<tr>
<td>Usability</td>
<td>3.3</td>
</tr>
</tbody>
</table>

asked to do an evaluation of the quality of the feature model, again using paired comparison. For each comparison, the stakeholders were asked which quality that they believed was more prominent in the feature model. The result was thus a ranking of what qualities that the resulting model had, as seen in Table 5.4. The ranks of the prioritization and the evaluation can then be compared (next section), thereby addressing Q2.

While doing the evaluation of the resulting model, the stakeholders were asked to indicate what properties, characteristics, or traits of the feature model that led them to believe that a quality took precedence over another, addressing Q3. This resulted in a matrix of properties of the feature model, versus the qualities seen in it. Knowing what indicators that promote or demote certain qualities, we can investigate how and why the indicators appear in a feature model. This leads us to derive guidelines for how to construct the model to promote or demote certain indicators, which in turn promote or demote certain quality factors. We can then use the guidelines to direct a feature model towards prioritized qualities, addressing Q4.

5.2.11 Comparison of prioritization and evaluation

The ranking of the qualities that the stakeholders saw as the most relevant in a model, and the evaluation of the resulting model, is shown in Figure 5.6.

Usability is clearly the quality that is most desired with reusability, changeability and correctness trailing close together. The highest ranked qualities that were found in the resulting model are usability and correctness. The largest difference between the desired and achieved quality is found in reusability, where the produced model is not reaching the preferred level.
Generally, the prioritized qualities are higher on ease of use and reuse at the expense of formalness and correctness. Most of the users are thus more interested in having a model that is oriented to practical and pragmatic qualities, rather than technically oriented, formal aspects. The resulting model, however, exhibit more of correctness and formal adherence. It is important to note that the evaluation was made on the model in itself, but that there could be confounding factors affecting the outcome, such as the user-friendliness of tools, etc.

There is also a more subtle factor to why the results of the evaluation leans towards the formal aspects of a model. It could be the case that the feature modeling method and the style of the models that results from feature modeling, inherently emphasize the qualities related to correctness and formality, while the expectation of the users of the feature models is to have a more pragmatic model of the components. While generative approaches to feature modeling are pursuing increased formalism, there appears to be a need and opportunity for research into the pragmatic side of modeling as well.

5.2.12 Indicators and interpretations

The evaluation procedure resulted in a set of comments on the qualities of the model and the basis for those qualities. The motivations and reflections of the

![Figure 5.6: Radar graph comparing the qualities that the stakeholders prioritize in a feature model, and the evaluation of the resulting feature model.](image-url)
stakeholders’ view on the quality of the model were analyzed, in order to find indicators in the feature model that signal the prominence of the particular quality factors. Five principal categories of indicators were found. Each of these categories can be further specified into measurable indicators and have metrics applied to them.

Based on the findings in the use case, we can list some examples of specific indicators related to the principal categories and their influence on the quality factors, seen in Table 5.5. Where an indicator leaves a positive contribution, the intersection with a factor has a raising contribution there is a filled symbol (●) and where there is a lowering contribution there is an empty symbol (○). If there is no, or an unclear influence, the space is left blank. A positive contribution means that prominence of the indicator in the feature model, implies increased prominence of the quality factor in the feature model.

### Supported modeling elements

Several quality factors of the feature model are affected by the meta-model used. The meta-model describes the modeling elements and constructs that are available for the modelers, and formalizes the model. The meta-model for feature models used in the case study (further detailed in section 6.1 and seen in Figure 6.2) is simple and straight-forward, but flexible. It provides the modelers with the possibility to define an arbitrary number of relationship types and feature attributes, but the validated feature model was quite constrained in this respect. Obviously, the presence of predefined types is only a concern if those attributes are strictly the only ones available, and the ability to define and use additional types is prohibited.

Having a simple and flexible meta-model was seen as a positive influence on the formalness of the feature model, since the modeling language is easy to learn when there are few modeling constructs to use and reduce the risk of creating models that violate the meta-model, since fewer modeling elements are easier to adhere to. On the other hand, the meta-model did not support different versions or revisions of the feature model elements, which was seen as detrimental to the changeability quality factor.

Examples of measures for this category of indicators are listed in Table 5.6. Apart from these indicators that were elicited from the case study, one can formulate further metrics for the feature modeling language or meta-model itself. For example, in [112] several characteristics are listed that describe the capabilities of ontology languages. The diversity in supported constructs is higher in ontology languages than in feature modeling languages, so we do not look into every
### Table 5.5: Examples of specific indicators found in the case study and their influence on the quality factors.

<table>
<thead>
<tr>
<th>Supported modeling elements</th>
<th>Changeability</th>
<th>Reusability</th>
<th>Formalness</th>
<th>Mobility</th>
<th>Correctness</th>
<th>Usability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version management present</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Few modeling elements</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Predefined relationship types</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Predefined feature attributes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Placement of features in hierarchies</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tree structure hierarchy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Predefined abstract features</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Predefined feature levels</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Structure and amount of dependencies</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Few hard relationships</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Few soft relationships</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Few relationship types</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Few features with relationships</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Amount of supplemental information</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Few feature attributes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Attribute commonality</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Overview and correspondence to actual artifacts</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Low average depth of hierarchy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ratio of feature/artifact mapping</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Low maximum depth of hierarchy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
possible difference between feature meta-models and their supported elements, but recognize that there can be meaningful differences.

**Placement of features in hierarchies**

The feature models produced, using the aforementioned meta-model, have a hierarchical, tree-like structure. During the case study there were no guidelines used on how to place and distribute features in the hierarchy. An alternative approach would have been to use any of the advice on feature distribution that exist in literature, or come up with original guidelines for the particular case. Recall from chapter 2 that there are several methods for feature modeling that suggests classifications for features. Some methods also provide general advice on criteria to use when placing features in the hierarchy. For example, [70] suggests that features should not be arranged in a functional decomposition, and that logical boundaries of the feature model should correspond to physical boundaries in the models of the architecture. [22] touch on aspects and their role in feature refinement. The general advice of [70] and others is to try to place features in a way that emphasize the variability in the model.

More freedom of choice on where in the hierarchy to place the features results in easier changes to, and updates of, the feature model. This has a positive impact on the changeability and reusability quality factors. The comprehensibility of the model is obviously highly influenced by the placement of the features in the feature model. If there are no prescriptions on where to place features that correspond to software artifacts, there is a greater chance that the features are placed where they are easiest to find, and therefore contribute to the usability of the feature model. However, the leeway could also result in the features being placed in abstractions and branches that result in semantic confusion.

Of course, sometimes the placement of the features is not clear, and the solution that results in the more comprehensible model might to a lesser extent reflect the circumstances in the modeled software artifacts. This results in negative contributions to the correctness of the feature model. During the validation activity of the case study, several features were found that could have been placed in different parts of the model, since the features had cross-cutting functionality. The final placement of the features was based on where it was perceived as easiest to find them.

The comprehensibility and ease of use of a feature model has often been attributed to the notation used. The aesthetics of the modeling notation and presentation format have not been considered in the case study. Presentation aspects of the feature distribution in the feature model, such as balance in the feature tree, etc., becomes either irrelevant or unappreciable when the model is of
### Table 5.6: Descriptions of indicators for supported model elements.

<table>
<thead>
<tr>
<th>Indicator name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version identification</td>
<td>In order to support some aspects of evolution and changes in feature models, it is necessary to support version management. Support for version control can either be present or absent from the modeling language used for the feature model.</td>
</tr>
<tr>
<td>Number of modeling elements</td>
<td>Just like parsimony in the model itself increases usability, parsimony in the language does the same. Fewer types of modeling constructs are quicker to learn and reduces the complexity of the modeling language.</td>
</tr>
<tr>
<td>Predefined relationship types</td>
<td>A multitude of different types of relationships and their semantics has been proposed over the years. While restricting the relationship types to a predefined selection of semantics may make the model conform to a particular notation, it can also limit the usability and the ability to model certain constructs. Predefined relationship types can either be present or absent from the model.</td>
</tr>
<tr>
<td>Predefined attribute types</td>
<td>Similar to relationship types, there are many suggestions for what types of attributes that can be associated with a feature. The restrictions imposed by predefined attribute types could be detrimental for the correctness of the model. Predefined attribute types can either be present or absent from the model.</td>
</tr>
</tbody>
</table>
reasonable size and there is a tool involved that provide browsing and searching functionality. The importance of – or lack thereof – notations for understandability of feature models was briefly discussed in section 3.5.1.

Examples of measures for this category of indicators are listed in Table 5.7.

### Structure and amount of relationships and dependencies

As seen in Table 5.3, there were few relationships and dependencies in the validated feature model from the case study. As could be expected, the evaluation of the model found that having few dependencies in the feature model made it easier to modify. Avoiding complex entangling of features via dependencies had a positive effect on the reusability. We do not discuss more elaborate metrics like fan-in/fan-out, or transitive dependencies here, since we did not find those issues in the case study.

The difference between dependencies and relationships is important in this case. There could be soft relations between features that are not dependencies, but of a more informative nature. Dependencies are a hard type of relations, where the structure of the dependencies affects how many features that are involved in each change.

The structure and nature of the dependencies in a feature model are closely related to where the features are placed, so the quality factors that are influenced the most are the same for structure of relationships as for placement of features.

Examples of measures for this category of indicators are listed in Table 5.8.

### Amount of supplemental information

A feature could have any type and amount of information attached as feature attributes of features or relations in the model (some examples are given in section 2.2.1). In the case study described here, the supplemental information was kept brief with only half a dozen feature attributes for each feature, though the meta-model used allowed the modelers to create any number of feature attributes. Reducing the amount of supplemental information to the essentials was found to be positive for the changeability, since it reduces the effort needed for making changes and adding new features to the model.

A limited amount of information was on, the other hand, also found to be negative for the correctness of the model, since it made it harder to convincingly determine whether the model was properly structured, when first encountering it. Relatively little information about the features also meant that it was harder to search the implemented feature model using keywords.

The larger scope that is intended to be covered by the feature model, the more features are included in the model, obviously. But in addition to the num-
Table 5.7: Descriptions of indicators for placement of features in hierarchies.

<table>
<thead>
<tr>
<th>Indicator name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tree structure hierarchy</td>
<td>If the structure of the features is expected and directed towards being in a tree structure, this influences the overall options available for where the features end up in the model. Since most of the metamodels for feature models that have been suggested use a tree structure, it improves the mobility quality factor to conform to a tree structure. It was also perceived as improving usability to have a tree structure that provides an outline to attach features and subfeatures to. Tree structure hierarchy can either be present or absent from the model.</td>
</tr>
<tr>
<td>Predefined abstract features</td>
<td>The placement of features in a model can be restricted – or guided – by the use of abstract features as categories or classifications of the features. These abstract features can be predefined or evolve as the model progresses. Having a predefined structure, which to some extent was present in the case study through the seeding, can improve correctness. The correspondence and agreement between other artifacts and models can be raised, but at the same time, the restrictions influence the changeability and reusability negatively. Predefined abstract features can either be present or absent from the model.</td>
</tr>
<tr>
<td>Predefined feature levels</td>
<td>Some feature modeling methods, notably FORM [62], prescribe feature models with several levels (Figure 2.6, p. 31). These levels, like predefined abstract attributes, influence the options available for positioning the features, but to a lesser extent. Using feature levels to describe the domains that a feature can belong to, can make it easier to understand the feature model, but predefined levels constrain changes and reuse. Predefined feature levels can either be present or absent from the model.</td>
</tr>
</tbody>
</table>
Table 5.8: Descriptions of indicators for structure and amount of relationships and dependencies.

<table>
<thead>
<tr>
<th>Indicator name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of hard relationships</td>
<td>The fewer relationships of hard types that exist in the feature model, the easier it is to change and reuse. It is not necessarily clear how the number of relationships affect the other quality factors. For instance, in the case of usability, a relationship might increase the understanding of how the model reflects other artifacts, but if there are too many they might be complex to overview. The amount of hard relationships is measurable as a number.</td>
</tr>
<tr>
<td>Number of soft relationships</td>
<td>The number of soft relationships in the feature model has the same effects as the number of hard relationships, except they are easier to disregard or override in case of changes or reuse. The amount of soft relationships is measurable as a number.</td>
</tr>
<tr>
<td>Number of relationship types</td>
<td>The semantics of the relations in a feature model determine how the features can be associated beyond the inherent hierarchical structure. Keeping the number of relationship types low reduces the amount of semantics in the model that needs to be understood, raising the understandability and usability quality factor. The amount of relationship types is measurable as a number.</td>
</tr>
<tr>
<td>Number of features with relations</td>
<td>The more features in the model that interact with other features, the more complex it becomes to overview and understand the implications of dependencies in the model. The amount of features with relationships is measurable as a number or a ratio. Which metric that is used depends on the context.</td>
</tr>
</tbody>
</table>
Table 5.9: Descriptions of indicators for amount of supplemental information.

<table>
<thead>
<tr>
<th>Indicator name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of feature attributes</td>
<td>Few attributes with supplemental information makes it easy to change the feature model, at the cost of correctness and some usability aspects. The amount of feature attributes can be measured as a number.</td>
</tr>
<tr>
<td>Attribute commonality</td>
<td>Having attributes that are in common for most features, reduces the risk of information overload and makes it easier to understand the contents of the model. On the other hand, it could mean that important information is missing from the feature model. The feature attribute commonality can be measured as a ratio between used and total number of attributes.</td>
</tr>
</tbody>
</table>

ber of features, the amount of supplemental information also grows in order to be able to accommodate the variety of information that the included features cover. In Table 5.9 we denote this “Attribute commonality”, in the sense that empty attributes for many features, i.e., attributes only used for a few features, clutter the feature model. In the case study, not all of the organization’s projects were included in the feature model. This was seen as beneficial to the usability quality factor, since the reduced information was easy to overview. Had more projects been included in the feature model, more attribute types had been required to represent the information about the features, and that could have made the model more unwieldy.

Examples of measures for this category of indicators are listed in Table 5.9.

Overview and correspondence to actual artifacts

The resulting model in the use case is quite shallow, in that it does not have deep hierarchical structures, and most features are found close to the top-level features. The short branches and relatively few variation points was found to benefit the usability of the model and provide good model overview. It also contributed to the changeability quality factor, since it was easy to find the appropriate place in the model to enter new features. Additionally, related to the placement of features in the model—the features were found to correspond well to the structure of the actual artifacts and assets that were modeled. The
Table 5.10: Descriptions of indicators for overview and correspondence to actual artifacts.

<table>
<thead>
<tr>
<th>Indicator name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average depth of hierarchy</td>
<td>If the average depth of the branches in the feature tree is kept low, the model is perceived as more usable and easier to overview. In order to keep the depth low, compromises with the accuracy of the features and artifacts might be needed, which would be negative for the correctness quality factor. The average depth of the feature hierarchy can be measured as a number or as a ratio of the maximum depth.</td>
</tr>
<tr>
<td>Maximum depth of hierarchy</td>
<td>Keeping the maximum depth of the feature hierarchy low makes the model easier to overview, improving the usability. If the features are kept closer to the top level features, they are also easier to move, which makes changes easier. The maximum depth of the hierarchy can be measured as a number.</td>
</tr>
<tr>
<td>Mapping between features and artifacts</td>
<td>Having artifacts to relate to the features in the model improves the understandability and the usability quality factor. The interoperability with other systems and tools is simplified if there is a mapping between the feature elements and the elements in other systems. The mapping between features and artifacts can be measured as a ratio of features that do not have an artifact counterpart.</td>
</tr>
</tbody>
</table>

agreement between features and corresponding software artifacts thus affects the usability quality factor, and was also considered to contribute to the mobility of the model, since the feature model could then be used for integration with other systems, models, or tools.

Examples of measures for this category of indicators are listed in Table 5.10.

5.2.13 Discussion of case study interpretations

Validity of indicators

There are several factors that could lead to the results found in the evaluation, and that are not related to the indicators mentioned. Among influencing factors
5. Initial Quality Model and its Application

there is the tool that was used, the source material in the form of documentation used as basis for the modeling, and the project context in the company. The only way to identify and attribute appropriate weight to such confounding factors is to design and carry out additional case studies, in order to gather more evidence from other settings.

There are a number of desirable properties that metrics should have according to ISO 9126 [54]. The quality metrics themselves should thus possess certain qualities:

**Reliability** so that random variations do not cause random errors in the results of the metric.

**Repeatability** so that repeated measurements on the same product in similar conditions by the same evaluator produce the same results, accounted for fatigue, learning effects, etc.

**Reproducibility** so that repeated measurements on the same product in similar conditions by a different evaluator produce the same results.

**Availability** indicating the conditions that constrain its usage.

**Indicativeness** so that the parts of the measured artifact that need improvement are identified.

**Correctness** so that the metric is objective, free of bias, and sufficiently precise.

**Meaningfulness** so that the metric produces meaningful results.

These properties are formulated for metrics of software quality, and are thus described as appropriate for measures on software. Although they are not stated with quality of models in mind, they provide a reference to evaluate metrics and indicators against.

Each of the indicators listed in the previous section satisfies these properties, and thus have valid descriptions and can be said to be appropriate for indicating the quality factors. The indicators are based on the results of a case study, and so are in a sense sound by design. If the indicators had been derived from theory alone, we would need to prove each property for each indicator. Since we now have empirical grounds for the indicators, we get confidence in the indicators’ validity for free.
5.2. Case study: Tactel AB

Modifiability of indicators

The indicators that were identified during the evaluation in the case study are quite intuitive and affect the quality factors in relatively predictable ways. However, the main point of interest is that the indicators that were found can be adjusted or modified within reasonable ranges. Thus, it is possible to affect the indicators and the quality factors and consequently also the quality of the feature model, so that the finished quality approaches that of the prioritized quality.

For the group of indicators in supported modeling elements, it is trivial to modify the indicators to affect the outcome of the affected quality factors by simply adding or removing support for the modeling elements. Likewise for the amount of supplemental information, which involves changing the number or structure of feature attributes. Also the placement of features in hierarchies are practically two-state boolean variables that can be changed, in order to make a tree structure hierarchy or predefined abstract features available. There is of course also the matter of the modelers using the abilities, once they have been made available to them.

The structure and amount of dependencies, and the overview and correspondence to actual artifacts are on the other hand not merely a matter of making abilities of the model available. It is considerably more complicated to modify the structure of the hierarchy and the relations in the model, compared to reducing the number of feature attributes in order to achieve higher attribute commonality. However, the experience of the modeling activities during the case study was that one can experiment with trade-offs between feature placement and dependency structure, meaning that it can very well be possible to affect the structure and amount of dependencies during the development of the model, and thus the quality factors that emerge as a result.

The amount of relationships in the feature model can be modified by considering how the features are distributed in the model. The need for a relation arises when two associated features are located in different branches of the feature hierarchy. Moving the features closer together in the hierarchy can thus reduce the need for relations in addition to the ones that are contained in the hierarchy. Incidentally, moving features closer would usually entail having the features closer to the top level features, and this would affect the average depth of the hierarchy which is another structural indicator. This thesis does not have a comparison matrix of how the indicators and quality attributes trade-off against each other, but this is an indication of how such interactions could arise.

The feature-artifact coverage is mainly a matter of scoping. The initial choices of which products to cover, of course, determine to a great extent which features that end up in the model. In the case study described here, there was
a natural choice for the abstraction level of the features that corresponded well to the artifact level. Depending on the intended usage scenario, a different abstraction level could be more suitable. An example would be that all software artifacts are categorized by functionality and each category represents a feature in the feature model. Conversely, the features could also illustrate variability below the most intuitive level of abstraction, which is exemplified in the case study by assigning features to the variability that is realized by parameterization of the plugins. The coverage and correspondence between features and software artifacts of different sorts is thus also possible to change in order to emphasize different quality factors.

**Validity of paired comparisons**

The consistency of a set of paired comparisons can be estimated by looking for circular triads. If a ranking of three items has an intransitive preference order like $k > j > i > k$, there is a lack of internal validity. Preferences and circular triads can be visualized using preference graphs. Figure 5.7 shows an example, taken from [23], of a preference graph with circular triads. An arc between two nodes indicates a preference in the direction of the arc. There are ten triads in all, and $ADB$ and $AEB$ are circular.

A double-sorted choice matrix can be used to detect circular triads, as described in [12]. Table 5.11 shows the resulting choice matrix for the priority ranking of one participant in the case study. A “1” in the table indicates that the quality factor in the column was preferred over the quality factor on the row. The upper and lower part around the principal diagonal are technically redundant to each other, but they assist in finding circular triads. Table 5.12 show the choice matrix after having sorted the raw choice matrix both by column score and row score. As can be seen, the upper and lower halves of the matrix are now...
5.2. Case study: Tactel AB

Table 5.11: One participant’s choice matrix for the prioritization of quality factors.

<table>
<thead>
<tr>
<th>Changeability</th>
<th>Reusability</th>
<th>Formalness</th>
<th>Correctness</th>
<th>Mobility</th>
<th>Usability</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Reusability</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>Formalness</td>
<td>1</td>
<td>1</td>
<td>-</td>
<td>1</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>Correctness</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Mobility</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>3</td>
</tr>
<tr>
<td>Usability</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Score</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 5.12: A double sorted choice matrix.

<table>
<thead>
<tr>
<th>Usability</th>
<th>Formalness</th>
<th>Reusability</th>
<th>Mobility</th>
<th>Changeability</th>
<th>Correctness</th>
<th>Usability</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>Correctness</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Changeability</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Mobility</td>
<td>0</td>
<td>0</td>
<td>-</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>Reusability</td>
<td>0</td>
<td>-</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>Formalness</td>
<td>-</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>Score</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

either 1 or 0, which means that there are no circular triads. This in turn means that the choices made by the respondent are internally consistent. Had they not been consistent, the halves of the matrix would have mixed values. All responses from the participants in the case study were analyzed using this approach. No circular triads were found in the prioritizations made, but there were four cases in the evaluation.
It is expected that the prioritization would lack circularity, since most people should be able to form an opinion about what qualities they find most important in the model and order them without ties. However, it is also reasonable that the evaluation can lead to circular triads—essentially ties—since there is nothing that stops two qualities from being equally prominent in the resulting feature model. It is perfectly legal for two or more quality factors to be equally prioritized or prominent in the evaluated feature model. There is no need to break ties in the evaluation stage, since the only way to do so would be to modify the model itself. High consistency and no ties in the prioritization is more likely, since the group of stakeholders would probably compromise around the preferred scenarios for the feature model. Like for the evaluation, internal consistency is however not a necessity.

There are other error sources that can affect the results of paired comparisons, just like any other type of evaluation. Examples of error sources would be small discrimination between alternatives, learning effects, order effects, etc. However, most of the error sources and their effects on the distributions on answers are not applicable in our case, since the context that the feature model should be used in is influencing the choices made.

**Predictive capabilities**

There are two capabilities that are often desired from a model and a method, perhaps particularly so when the model concerns quality. The first is descriptive capability, which is what is described so far in this thesis, in that the model can describe the quality of the resulting product along the quality factors, based on the indicators. The second is predictive capability, where the model can make a prediction of how the quality factors are affected by changes. While the previous paragraphs described how changes of the indicators can be used to affect the quality factors, the model cannot be used to make a prediction of just how prominent one quality factor becomes at the expense of others when a change is made. The indicators are found in a qualitative evaluation and do not have quantitative connections to the quality factors yet. That is, we do not know if a small change of an indicator results in a small change of the quality factors, and we do not know where the break points are when further changes of the indicators do not result in further effects on the quality factors.
5.3 Example values of the indicators

This section exemplify each of the indicators by the feature model that resulted from the Tactel case study. Table 5.13 illustrates the indicators by showing the values from the final model of the case study.

Table 5.13: Examples of specific indicators found in the case study and their values.

<table>
<thead>
<tr>
<th>Indicator</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Supported modeling elements</strong></td>
<td></td>
</tr>
<tr>
<td>Version management present</td>
<td>Present</td>
</tr>
<tr>
<td>Few modeling elements</td>
<td>6</td>
</tr>
<tr>
<td>Predefined relationship types</td>
<td>Absent</td>
</tr>
<tr>
<td>Predefined feature attributes</td>
<td>Present</td>
</tr>
<tr>
<td><strong>Placement of features in hierarchies</strong></td>
<td></td>
</tr>
<tr>
<td>Tree structure hierarchy</td>
<td>Present</td>
</tr>
<tr>
<td>Predefined abstract features</td>
<td>Absent</td>
</tr>
<tr>
<td>Predefined feature levels</td>
<td>Absent</td>
</tr>
<tr>
<td><strong>Structure and amount of dependencies</strong></td>
<td></td>
</tr>
<tr>
<td>Few hard relationships</td>
<td>11</td>
</tr>
<tr>
<td>Few soft relationships</td>
<td>0</td>
</tr>
<tr>
<td>Few relationship types</td>
<td>3</td>
</tr>
<tr>
<td>Few features with relationships</td>
<td>0.06</td>
</tr>
<tr>
<td><strong>Amount of supplemental information</strong></td>
<td></td>
</tr>
<tr>
<td>Few feature attributes</td>
<td>10</td>
</tr>
<tr>
<td>Attribute commonality</td>
<td>0.49</td>
</tr>
<tr>
<td><strong>Overview and correspondence to actual artifacts</strong></td>
<td></td>
</tr>
<tr>
<td>Low average depth of hierarchy</td>
<td>2.7</td>
</tr>
<tr>
<td>Ratio of feature/artifact mapping</td>
<td>0.86</td>
</tr>
<tr>
<td>Low maximum depth of hierarchy</td>
<td>5</td>
</tr>
</tbody>
</table>
5. Initial Quality Model and its Application

5.4 Chapter summary

This chapter has described the first version of a quality model for feature models and its application in a case study with a developer of mobile applications. The initial model proposes a set of quality factors, divided into quality attributes. The quality factors and quality attributes can be evaluated by using indicators that form measurable and evaluable properties of a feature model. The initial quality model is based on the practices and approaches that exist for other software models and software artifacts.

In the case study, an SME that operates in the domain of mobile applications is described. Mobile applications are very variability demanding, and the case study describes the development of a feature model for an in-house software component framework. Paired comparisons for prioritization and evaluation of quality factors were used during the development of the feature model. The instrumentation, analysis, and management of the paired comparisons approach were simple and straightforward, making the approach practical and accessible.

A web-based modeling tool called FMWIKI was used for modeling. While there are many potential improvements that can be made to the tool, it proved useful and solved the issues of visualization and concurrent, distributed model management in an elegant manner.

Based on the quality of the produced feature model, indicators for feature model quality factors were found and described. The indicators can be influenced in order to promote certain qualities in the feature model.
Refined Quality Model and Approach

The application of the initial quality model in the second use case resulted in validation for some aspects of the quality model, while it also resulted in modifications to other parts. In this chapter, a second iteration of the quality model is presented with the experiences from the case incorporated. This refined version of the quality model has a stronger basis, since it is designed not only from theoretical constructs, but also formed by practice.

This chapter also describes the procedures and instrumentation that were used in the applications for prioritizing what qualities that are most important, how the results of prioritizing affects the feature model development, how to evaluate the results, and proposes how to integrate and use the quality model in feature model development practice and variability management.

6.1 Second iteration of the quality model

Figure 6.1 shows an overview of the elements comprising the refined quality model for feature models. The most essential changes of the revised method, compared to the initial approach (Figure 5.1, p. 107), is that the quality attributes have been removed, the measurement has been split into prioritization and evaluation, and the approach now emphasize an iterative nature, while the initial approach ended in a result. Furthermore, some elements have been renamed to better reflect the activity carried out, or the artifact that is produced.
The reason for removing the quality attribute element from the approach is that there is no empirical material to support the use of such specializations of the quality factors. The essential values of the quality attributes can still be captured under the quality factors. The following list explains the elements of the refined method.

**Scenario**  The intended usage scenario of the feature model is the basis of the prioritization made by the stakeholders. Given different scenarios, different quality factors could be more relevant.

**Constraints**  Different aspects of the context that the feature modeling takes place in, influence the development of the model. The development principles that can be used and applied during the development are constrained by the context. For example, project resources, business domain, technical constraints, etc., can all place restrictions on how the development of the feature model can progress.

**Quality factor**  The quality factors embody the main qualitative properties that can be prominent in a feature model. During prioritization, the stakeholders of the feature model identify and rank the qualities that should be dominant in the finalized feature model.
Prioritization  Given the intended use of the feature model, the stakeholders of the feature model express what qualities that are the most important for the model to possess. The prioritization determines which development principles that should be recommended to abide by during the modeling.

Development principle  Depending on what priorities that are given to the quality factors, there are different development principles that contribute to increasing the prominence of the different qualities. The development principles are based modifying on the indicators that point out properties of the feature model. The principles are validated during the evaluation of feature models that the principles are used in.

Evaluation  Once the feature model has been developed, the stakeholders evaluate the final model and appraise the prominence of the different quality factors. This activity finds evidence in favor of, or opposing, the use of development principles to strengthen the prominence of particular qualities in the feature model.

Comparison  The prioritization and the evaluation are to be compared in order to find the indicators of certain quality factors. The comparison detects or refines the indicators of certain quality factors in the feature model. Over time, the comparison of the priorities, and thus the used development principles, and the result will refine the indicators and their effect on the model.

Method of measurement  Different types of indicators require different measures to capture them, and the nature of the method of measurement used influences what kind of indicators that are available for a quality factor. The measurements are applied during the evaluation of a feature model, in order to find the indications of certain quality factors.

Indicator  Actual measurement or evaluation of a quality factor is performed on indicators. Each quality factor can have several indicators. An indicator for the prominence of a quality results in one or more development principles for feature models. The indicators provide a measurable value of a property in the feature model that can be modified in order to emphasize, or suppress, the prominence of an associated quality.

There are many choices for meta-models when it comes to feature models. We present the meta-model for feature models that we have used in the case study, for completeness (Figure 6.2). It allows the modelers to define attributes and relations to be attached to features in a very flexible manner, and is easily
implemented in a relational database. The default feature hierarchy is tree-formed, and other relations are solved by adding further relationship types. The default relationship in our models’ hierarchical structure is optionality.

A feature is part of a feature hierarchy, where each feature has a parent feature. The model does not allow for more than one parent per feature, but pragmatically there is no need to accommodate such a construct. Any such needs can be solved using suitable dependency relations, while the ensuing tree structure of the current meta-model is easy to understand. Considerable effort has been spent in feature modeling research trying to determine what types of relations that should exist in feature models, and also what additional information about a feature that is meaningful to store. The meta-model used in our work allows the modelers to create any type of attribute describing a relevant property with a feature, while not presenting a fixed set of possible relations. Since there are few publicly presented cases in variability modeling where the situation of the modeling organization has been fully accommodated using the same setups as previous projects, it makes more sense to avoid imposing restrictions.

A feature in our case can come in various types. Using terminology used by other authors, the features can be:

- abstract, indicating a concept that must be realized by a sub-feature in the form of a plugin. Principally, this is a categorization,
- concrete, which in our case study means a plugin,
- collection, that acts as a container for a feature group,
- parameterization, which in our case study almost always is realized as a parameter to a plugin.

Out of the quality factors that were part of the initial model, a refinement has been made and six quality factors have been identified and slightly redefined.
6.1. Second iteration of the quality model

The basis of this choice of quality factors is the experiences made in the case study. The practitioners found the quality factors to be the most important and embody the essential properties to be found in a feature model. The name of the quality factor that was called veracity in the initial model has been changed to correctness, which was seen as an easier name to use.

**Changeability** is the ability of the model to evolve as the modeled product line evolves, while maintaining the same purposes and applications as previous iterations of the model. A model describing a product line that is expected to change rapidly or regularly must have the flexibility to accommodate those changes without requiring too extensive efforts for re-design.

**Reusability** encompasses the properties of the model necessary to enable or facilitate the reuse of the current model when evolving or developing other models. The level of reuse can be on different levels of granularity, such as an entire model or partitions of it.

**Formalness** is related to the ability to manage the model in a formalized manner, notably for the purpose of machine management, tool support, and automation. Conformance to notations, redundancy, and consistency are examples that may affect the level of formalness in a model.

**Mobility** concerns the model’s ability to be installed and made available to the users, integrated with tools, interact with existing processes, moved or duplicated to different systems or tools.

**Correctness** is the quality of the models veracity and appropriateness for the modeled scope. The accuracy in modeling the real artifacts, the level of completeness and coverage of the modeled domain, and the reliability of the usage of the model resulting in accurate solutions are sub-qualities of the correctness. While having a correct model would seem to be a top priority, it could also mean that the learnability, reusability, or changeability are negatively affected.

**Usability** is the model user’s ability to employ usage and receive beneficial effects of the model. A usable model should be easy to learn, accepted by the stakeholders as a suitable representation, accessible and visible to the users, and understandable for the stakeholders. In order to reach high usability one might have to sacrifice correctness and formalness.

During the case study, a number of indicators were identified for the quality factors and are now part of the refined quality model. Section 5.2.12 describes and discusses the indicators found during the case study.
For now, we do not have enough material to say much about the relations between quality factors. We expect certain factors to increase or decrease their prominence in concert with modifications of other factors. We can review the indicators found and their positive and negative contributions to the quality factors (Table 5.5) and based on those contributions deduce how some quality factors seem to co-vary, either in the same or opposite directions. Table 6.1 shows the result of that analysis. Two quality factors that appear to positively co-vary have a ◊ in the intersection, while quality factors that seem to preponderate at the expense of each others have ⊙ in the intersection. If the relation between two quality factors is unclear, the intersection is left blank.

### 6.2 Prioritizing the quality of feature models

This section describes the prioritization component of the quality model process. The prioritization is performed by the stakeholders of the feature model and is an input for the construction phase. The analysis of the prioritization results in a recommended set of guidelines that should be used during the construction of the feature model. The input to the prioritization is the scenario that the feature model is expected to function in, and the quality factors that are to be compared.

#### 6.2.1 Scenario

Scenario is the term used to describe the purpose and intended use of the feature model under development. As stated previously, there are many possible uses for feature models, and we found in the first case study that stakeholders’ visions regarding the model’s use result in different models. The formulation of the scenario is influenced by various aspects, such as a long-term strategy for

<table>
<thead>
<tr>
<th>Table 6.1: Relations between quality factors.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Changability</td>
</tr>
<tr>
<td>Reusability</td>
</tr>
<tr>
<td>Formalness</td>
</tr>
<tr>
<td>Mobility</td>
</tr>
<tr>
<td>Correctness</td>
</tr>
<tr>
<td>Usability</td>
</tr>
</tbody>
</table>
6.2. Prioritizing the quality of feature models

products, or particular problems that result from uncontrolled variability. The scenario is instrumental in setting the scope for the feature model, finding the appropriate stakeholders of the model, and provide the basis for the prioritization.

Depending on the envisioned usage scenario for the feature model, different quality factors can be seen as more or less important to the stakeholders. The scenario is also influenced by the context or situation that the modeling organization is in. Examples of how the organization's situation can influence the scenario and the desirable model qualities are:

• the organization is in a situation that leads to a scenario with frequent restructuring or refactoring of the software portfolio, and would like to emphasize the importance of reusability,

• the organization is in the initial stages of variability modeling, and require foremost a model that is easy to understand and learn, in order to get feedback from as many as possible, or

• the organization has frequent changes in tool chains and other development resources, that require high mobility from the feature model in order to (re-)integrate it.

Given that there is an effort and a cost involved in producing a feature model, the scenario would probably be guided by the principle that it should provide benefits for as many as possible. However, one should keep in mind that variability management could be performed on different levels (e.g. product, problem, solution), and the most appropriate scenario might not be the one that influence the majority of the organization.

6.2.2 Procedure

The procedure for prioritizing the quality factors in the feature model is based on the paired comparison method. The format should be subject to tailoring if it cannot be applied directly in a useful manner. Each stakeholder compares the different pairs of the six quality factors and chooses the quality that they think is more important for the feature model to possess.

While it could be beneficial to have the stakeholder group negotiate the priority of different qualities, and reach an agreement on the order, it is an activity that could require lengthy and resource consuming meetings of all the stakeholders. By using a simple paired comparison method, the prioritization can be carried out in a distributed manner. When performing the prioritization of the quality factors it is suitable to randomize the order of the paired comparisons
to avoid systematic bias. There could be learning effects related to the comparisons if they follow a systematic pattern. A randomized order also helps to distribute effects of tiredness or stress across all comparisons when making the prioritization.

Unless the stakeholders are already aware of the meaning of the quality factors, it is necessary to begin the prioritization with a quick introduction to the quality factors. Stakeholders are likely to have opinions regarding the definitions and descriptions of the quality factors. In order to maintain consistency during the distributed prioritization it is important that all stakeholders work with the same definitions. Changes to the definitions should thus be made prior to the prioritization. Although the definitions should be the same for all, there will always be room for interpretation on behalf of the respondents to the prioritization. That interpretation and the stakeholders’ own understanding of the definitions is due to the influence of the scenario for the feature model, and is thus in order and not controlled.

It is not necessary to avoid or break ties in the prioritization. It is valid to have two or more quality factors receive the same priority. In many other cases were paired comparisons are used there is in fact a difference between all choices that are made, but when it comes to quality factors there is no true objective order of the options. All the choices are made subjectively and are intended to reflect the subjective opinion of the stakeholders. Should a special case appear and breaking ties would become necessary, there are different ways to do so found in literature, such as looking at wins between the tied options or flipping a coin [23].

6.2.3 Instrumentation

An example of the form used for performing the paired comparison in the case studies is seen in Figure 6.3. Note that the order of the comparisons seen here is not randomized. In the case study described in the previous chapter, the form was printed and filled out manually by the stakeholders. The form could easily be administered through a web page or integrated with tools.

The form has shortened definitions for the quality factors, as seen in the figure and listed below.

**Changeability**  Ability to evolve the model while maintaining the uses of previous versions.

**Reusability**  Ability to reuse (parts of) the model when evolving or developing other models.
## Prioritizing the quality of feature models

### Description

| Ability to evolve the model while maintaining the uses of previous versions | Changeability | Re-usability | Correctness | Mobility | Usability |
| Ability to evolve the model while maintaining the uses of previous versions | Changeability | Formalness | Correspondence (mapping) between the model and the modeled artefacts | Ability to be moved, transferred and integrated with other systems | User-friendliness and ease of learning and communication to new users |
| Ability to evolve the model while maintaining the uses of previous versions | Changeability | Correctness | Ability to be moved, transferred and integrated with other systems | Usability | Communication to new users |
| Ability to manage the model in a formalized manner, e.g. for machine management | Formalness | Correctness | Correspondence (mapping) between the model and the modeled artefacts | Mobility | User-friendliness and ease of learning and communication to new users |
| Ability to manage the model in a formalized manner, e.g. for machine management | Formalness | Mobility | Ability to be moved, transferred and integrated with other systems | Usability | Communication to new users |
| Ability to manage the model in a formalized manner, e.g. for machine management | Formalness | Usability | User-friendliness and ease of learning and communication to new users | Mobility | Correspondence (mapping) between the model and the modeled artefacts |
| Ability to manage the model in a formalized manner, e.g. for machine management | Formalness | Correspondence (mapping) between the model and the modeled artefacts | Ability to be moved, transferred and integrated with other systems | Usability | User-friendliness and ease of learning and communication to new users |
| Ability to manage the model in a formalized manner, e.g. for machine management | Formalness | Ability to be moved, transferred and integrated with other systems | Usability | Correspondence (mapping) between the model and the modeled artefacts | User-friendliness and ease of learning and communication to new users |
| Correspondence (mapping) between the model and the modeled artefacts | Correctness | Mobility | Ability to be moved, transferred and integrated with other systems | Usability | User-friendliness and ease of learning and communication to new users |
| Ability to be moved, transferred and integrated with other systems | Mobility | Usability | User-friendliness and ease of learning and communication to new users | Correspondence (mapping) between the model and the modeled artefacts | Correctness |

### Quality

<table>
<thead>
<tr>
<th>Changeability</th>
<th>Re-usability</th>
<th>Correctness</th>
<th>Mobility</th>
<th>Usability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Formalness</td>
<td>Correspondence (mapping) between the model and the modeled artefacts</td>
<td>Ability to be moved, transferred and integrated with other systems</td>
<td>Usability</td>
<td>Correspondence (mapping) between the model and the modeled artefacts</td>
</tr>
<tr>
<td>Ability to manage the model in a formalized manner, e.g. for machine management</td>
<td>Formalness</td>
<td>Correctness</td>
<td>Mobility</td>
<td>Usability</td>
</tr>
<tr>
<td>Ability to manage the model in a formalized manner, e.g. for machine management</td>
<td>Formalness</td>
<td>Mobility</td>
<td>Usability</td>
<td>Correspondence (mapping) between the model and the modeled artefacts</td>
</tr>
<tr>
<td>Ability to manage the model in a formalized manner, e.g. for machine management</td>
<td>Formalness</td>
<td>Usability</td>
<td>Mobility</td>
<td>Correspondence (mapping) between the model and the modeled artefacts</td>
</tr>
<tr>
<td>Ability to manage the model in a formalized manner, e.g. for machine management</td>
<td>Formalness</td>
<td>Correspondence (mapping) between the model and the modeled artefacts</td>
<td>Usability</td>
<td>Mobility</td>
</tr>
<tr>
<td>Ability to manage the model in a formalized manner, e.g. for machine management</td>
<td>Formalness</td>
<td>Ability to be moved, transferred and integrated with other systems</td>
<td>Usability</td>
<td></td>
</tr>
</tbody>
</table>

### Usability

| Ability to evolve the model while maintaining the uses of previous versions | Changeability | Re-usability | Correctness | Mobility | Usability |
| Ability to evolve the model while maintaining the uses of previous versions | Changeability | Formalness | Correspondence (mapping) between the model and the modeled artefacts | Ability to be moved, transferred and integrated with other systems | User-friendliness and ease of learning and communication to new users |
| Ability to evolve the model while maintaining the uses of previous versions | Changeability | Correctness | Ability to be moved, transferred and integrated with other systems | Usability | Communication to new users |
| Ability to manage the model in a formalized manner, e.g. for machine management | Formalness | Correctness | Correspondence (mapping) between the model and the modeled artefacts | Mobility | User-friendliness and ease of learning and communication to new users |
| Ability to manage the model in a formalized manner, e.g. for machine management | Formalness | Mobility | Ability to be moved, transferred and integrated with other systems | Usability | Communication to new users |
| Ability to manage the model in a formalized manner, e.g. for machine management | Formalness | Usability | User-friendliness and ease of learning and communication to new users | Mobility |
| Ability to manage the model in a formalized manner, e.g. for machine management | Formalness | Correspondence (mapping) between the model and the modeled artefacts | Usability | Mobility |
| Ability to manage the model in a formalized manner, e.g. for machine management | Formalness | Ability to be moved, transferred and integrated with other systems | Usability | Correspondence (mapping) between the model and the modeled artefacts |
| Correspondence (mapping) between the model and the modeled artefacts | Correctness | Mobility | Ability to be moved, transferred and integrated with other systems | Usability |
| Ability to be moved, transferred and integrated with other systems | Mobility | Usability | User-friendliness and ease of learning and communication to new users | Correspondence (mapping) between the model and the modeled artefacts |

### Figure 6.3: Sample of form used for prioritization of quality factors.
Formalness  Ability to manage the model in a formalized manner, e.g. for machine management.

Correctness  Correspondence (mapping) between the model and the modeled artifacts.

Mobility  Ability to be moved, transferred and integrated with other systems.

Usability  User-friendliness and ease of learning and communication to new users.

The stakeholders have access to the full definitions of the quality factors, and the shortened definitions mainly serve to remind the participants in the prioritization of the gist of the definitions. Provided that the stakeholder has strong interests in the feature model under consideration, and also sufficient experience, the instrumentation could be expanded to involve not only the top level quality factors, but also the quality attributes listed in section 5.1.2. However, the number of comparisons for 25 attributes is $\binom{25}{2} = 300$ instead of 15 comparisons for 6 quality factors. It would probably be more suitable to use a second step, where only the highest ranked quality factors are investigated on a more detailed attribute level.

Calculations

The only calculations necessary are some summations of the number of times that one quality is preferred over another, and can easily be done by hand or using common software.

Let $\{q_1, \ldots, q_6\}$ be the quality factors. Let $x_{ij}$ where $i, j = 1, \ldots, 6, i \neq j$ assume 1 or 0 respectively if $q_i$ is preferred over $q_j$ or if $q_j$ is preferred over $q_i$. Let $|q_k|$ where $k = 1, \ldots, 6$ be $\sum_{j=1, j \neq k}^{6} x_{kj}$, i.e. each quality factor $q_i$ was preferred $|q_i|$ times.

The order of priority for each individual judge $a$ is then ordered on ($|q| \geq$), i.e. the quality factors sorted according to how many times the judge preferred them.

The order of priority for the group of stakeholders $A = a_1, \ldots, a_n$ is ordered on ($\sum_{i=1}^{n} |q_{a_i}| \geq$), i.e. the quality factors sorted according to how many times the judges combined chose them. Dividing by the number of judges gives the average number of times that each quality was chosen by the group.
6.3 Evaluating the quality of feature models

The evaluation of the quality of the feature model uses the same instrumentation as the prioritization. For each paired comparison, the evaluator decides which quality factor that is more prominent in the model. Once the evaluation has been made, the same simple calculations as for prioritization are made, and the resulting order of the quality factors from the prioritization and the evaluation can be compared.

The main difference between the prioritization and the evaluation is that the evaluation involves not only selecting which quality that the stakeholder believe is more prominent, but also motivating why it is more prominent. Through these motivations, candidates for further indicators of the quality factors are detected and collected. Over time, this also gather evidence to support or reject that indicators are correctly identified and attributed to the different quality factors. The motivations can range from being very general to very specific comments, and in order to develop an indicator, the comments need to be analyzed and a metric constructed for the indicator. The indicators will in turn give rise to development principles, either new or refinements of existing.

As already mentioned, the evaluation of the resulting feature model can be sensitive to several factors, such as the notation and the tool used for developing the feature model. It is important to distinguish between evaluating the variability in the feature model and evaluating the representation or presentation of the model. If possible, the influence on the results of the evaluation that are due to these factors could be somewhat mitigated by letting the model be evaluated in different notations and distribute them randomly in the stakeholder group. This is however an advanced measure that will likely not be feasible in most cases.

Even though the analysis of the motivations and indicators for the quality in the feature model should differentiate between the concerns of the model as such and its presentation or representation, this does not mean that the information about the latter aspects should be discarded. Comments regarding the presentation and representation of the model can provide feedback that may help improve for example tools and notations, so that the influence of such factors on the perceived feature model quality is reduced over time.

6.4 Comparison

Given the results of the prioritization and the evaluation of the finished feature model, we can compare the anticipated outcome with the actual outcome. The comparison can be visualized using for example a graph, such as that seen in Figure 5.6. At the moment it is only the order of the quality factors that can
provide information regarding the prominent qualities in the feature model. The value of the quality factors are on an ordinal scale, rather than a nominal scale. It is thus not possible to conclusively say that a quality is twice as prominent than another, if the average number of choices for one quality factor is twice that of the number of choices for another quality factor. Nor is it possible to say that the prominence of a quality can be raised by a certain amount by performing certain changes to the feature model. If information is gathered from sufficiently many feature models and evaluations, such mechanisms may be possible to formulate in the future.

The comparison provides a hint of whether or not the quality goals for the feature model has been achieved or not. A hint, since the quality goals of the feature model might have been reached to a sufficient extent, even if the order of the most prominent quality factors and the highest anticipated quality factors is different. If the difference in how the feature model turned out and what was expected is small, then the model is likely to be accepted by most stakeholders. If the discrepancy is large however, it may be necessary to rework the feature model and make a new revision of it in order to better meet the quality goals.

Taking the result from the Tactel case study as an example, the prioritization and the evaluation are not fully in line with each other. The major discrepancy is on the reusability quality factor, while the other factors are within 0.7 choices each on average. Since the feature model produced was the first feature model in the organization, a lack of reusability was not a sufficient reason to make any major revisions of the feature model.

The use of paired comparison in this context is deliberately kept simple to understand and perform. The arguably most popular method involving paired comparisons and evaluations is the Analytical Hierarchy Process (AHP) as described by Saaty in many publications [101]. AHP involves making not only paired comparisons, but also determine the relative weights of the options being compared, some more complicated calculations and arriving at a result where one can state that one or other choice is more or less prioritized by a factor. While the computations of AHP are usually conveniently handled by tools, the comparisons and weighting still add some extra effort to the activity. In order to make appropriate judgements on how much more or less important quality factors are over others, a stakeholder would need quite a bit of experience. Compared to selecting, for instance, how much more important fuel efficiency is over the looks of a car, quality factors of feature models are certainly more demanding.
6.5 Constraints

Development of the feature model will invariably take place under some constraints. While we previously mentioned “scenario” as something that could be influenced by the organization, in order to achieve certain goals, the constraints are aspects that are typically outside of general control.

The constraints on the prioritization can in part be due to effects of the domain of the organization. In requirements engineering and systems development, it is commonly argued, that a safety-critical business domain requires the processes and artifacts to be validated and verified in a manner that is more strict than for some other domains. Feature models are often a part of a software product line approach and can be used to describe compositions of software and system components. Should the components and compositions be required to demonstrate e.g. safety or security in a formal manner, it could also be the case that the model describing these would require higher degrees of correctness and formality in order to allow formal checks of feature interactions.

Other aspects of context that could affect the feature model development was mentioned previously, such as available modeling resources and technical constraints. The available resources obviously limit, for example, the number of iterations that can be undertaken in order to reach the quality goals. Technical issues could pose limits through the meta-models available or the sources for detecting features that can be used. Examples from the case study described in this thesis are the availability of staff for the projects comprising the feature model, and how sensitive information in parts of the documentation associated with the feature model constrains how the model can be distributed.

6.6 Development principles

Given the ranking of what quality factors that are the most important for a feature model, the goal is of course to promote those qualities in the model. The idea behind development principles is to use a preventive approach to the quality in feature models, rather than detecting potential problems at a later stage. Using the connection between quality factors and indicators, it is possible to take steps already in the construction and modeling phases to increase the prominence of certain qualities in the resulting model, thereby reducing the need for reiterations and changes of the model after completion.

Since the indicators of the quality factors are related to actual properties of the feature model, it is possible to reinterpret the indicators into development principles that are more approachable during modeling activities, whereas the indicators are suitable for evaluation of the resulting model. The word principle
6. Refined Quality Model and Approach

Table 6.2: The influence of development principles on the quality factors.

<table>
<thead>
<tr>
<th>Development Principle</th>
<th>Changeability</th>
<th>Reusability</th>
<th>Formalness</th>
<th>Mobility</th>
<th>Correctness</th>
<th>Usability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Principle of language parsimony</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Principle of strict typing</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Principle of subsumption</td>
<td>○</td>
<td>○</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Principle of vertical domains</td>
<td>○</td>
<td>○</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Principle of dependency parsimony</td>
<td>●</td>
<td>○</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Principle of information parsimony</td>
<td>●</td>
<td>○</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Principle of artifact agreement</td>
<td>○</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Principle of shallow trees</td>
<td>●</td>
<td>○</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

should be interpreted as recommendation. In the vein of existing guidelines for increasing the quality of other model types, such as [104], the development principles should embody advice and best practices for reaching the intended quality goals.

This section formulates some development principles from the indicators that were found in the Tactel case study. The principles are short formulations that provide an instruction, the anticipated effect of the instruction, and a brief rationale for using the principle. The development principles to promote quality factors are complementary to other, general, guidelines for feature modeling such as [70]. A general approach for feature modeling that was used in the case study is found in [116]. The principles listed here map nearly one-to-one to the indicators mentioned previously. As more indicators are found and formulated, either more principles would be induced, or existing principles would be updated or modified.

Table 6.2 shows the familiar matrix of quality factors and the influence that the different development principles have on the quality factors. Adhering to the development principles marked with ● promotes the quality, while the principles marked ○ demote the quality.
Principle of language parsimony

The feature modeling language used during the modeling should be economical in how many constructs that it offers to the modelers. In order to reduce the complexity of the feature model and reduce the effort required to learn the model, an evaluation should be done by the modeling leaders and the modeling team, considering what elements that are necessary to use from the rich variety of feature meta-models that are available. This should consider feature types, relationships, grouping, and labeling of elements.

Principle of strict typing

In order to reduce the complexity of the resulting feature model, the feature types, relationship types, and feature attribute types for the features should be defined before the modeling commences. The allowed types of features, relations, and attributes should either be selected based on the needs for the current model, or adhere to a predefined organization standard. Such a standard could be taken from an existing feature modeling practice, a de-facto methodology, or be the result of previous feature modeling efforts within the organization. Adherence to such a standard with predefined types, ensures compatibility between feature models.

Principle of subsumption

The features should be organized in a hierarchy with a subsumption structure. Essentially, this means that a tree structure should be used rather than a general directed graph. All features except the root feature should have a parent feature, that must be included in order to include the features that the parent subsumes. Each feature should only have one parent feature. Complex feature interactions that require cross-level relations should be resolved by using dependencies.

Principle of vertical domains

Features should be categorized and organized in vertical domains. Each feature that is added to the model should be belong to a classification that has been predefined. Examples of vertical domains are those found in for example FORM [62], namely capability layer, operating environment layer, domain technology layer, and implementation technique layer. The actual vertical domains used should be adapted to the specific organization. The use of vertical domains emphasize and encourage reuse and separation of concerns on the variability level.
6. Refined Quality Model and Approach

**Principle of dependency parsimony**

Relations in general, and dependencies in particular, should be used sparingly. The features should to the extent possible be organized so that all important interactions between features in the model are realized through the feature hierarchy, thus avoiding the need for further coupling. The types of relationships available for use in the model should be kept low. The number of features that have dependencies associated with them should be kept low. For example, it might be possible to associate parent features through a modified relation, instead of each of the sub-features having individual relationships to the same target or from the same source. Depending on the meta-model used, and the nature of the modeled domain and scope, more detailed principles may be applicable, such as minimizing the number of mutually exclusive features.

**Principle of information parsimony**

The number of feature attributes that provide supplemental information and details about the features should be kept low. The attribute selection should start by finding the smallest set of attributes that is common to all features in the modeled domain or scope. Attributes that do not provide useful information for all the features should be used sparingly in order to reduce redundant information in the model.

**Principle of artifact agreement**

Features in the feature model should to the greatest extent possible have a functional mapping to a software or system artifact. Features that do not correspond to an artifact should be clearly labeled as being abstract or unimplemented. Examples of artifacts are classes, modules, functions, software or hardware components, documentation or design artifacts. The semantics of the feature hierarchy should correspond to a hierarchy or classification that is visible in the modeled artifacts. Relations between features should have a corresponding relation realized between the modeled artifacts.

**Principle of shallow trees**

The maximum and average depth of the feature hierarchy should be kept low. In order to reduce the depth of the feature tree, the upper level features can be made abstract or used as classifiers. This permits the upper branches to be more generic categories or have a wider focus that can contain a wider range of sub-features. Reordering features in the tree so that intermediate abstract features can be removed reduces the depth.
6.7 **Integration and operation**

The previous sections have described the various elements that are part of the proposed quality model for feature models. This section describes how the quality model interacts with other procedures and processes, specifically already existing and established feature modeling methodologies and techniques, and the effects that the use of the model has on the general development process, with a focus on the particularities of SMEs.

6.7.1 **The quality model and the feature modeling process**

A fundamental premise of this thesis is that there are means and measures that can be taken to increase the quality of feature models compared to the existing approaches. However, this does not mean that the existing work on feature modeling methodologies and techniques is dismissed as inferior or unsuitable for the purpose of variability modeling. The consideration of quality aspects is intended to improve on the existing practices, as an extension of the capabilities rather than a non plus ultra replacement of the methods. To accomplish this, the quality consideration should be possible to integrate with existing processes. In order to demonstrate this, the methods for feature modeling that were described in section 4.1 are reviewed with regard to how they are compatible with the quality model.

The comparison and evaluation of compatibility is made on the basis of the development principles. The development principles can to various degrees be either already present and encouraged by the method (●), compatible with the method (○), or incompatible or discouraged by the modeling method (□). The result is summarized in Table 6.3 and briefly commented below.

**FORM**

The FORM-notation for feature modeling has relatively many predefined elements, in particular in the form of a feature hierarchy that has several different meanings for the subsumption hierarchy. It prescribes a number of feature types and relationship types, but can accommodate extensions and a variety of feature attributes. FORM makes use of vertical domains and predefined levels to order the features. The use of vertical levels results in detailing features to a more specific level from more generic functionality down to implementation methods. This results in comparatively deep feature hierarchies.
**Table 6.3:** The compatibility between proposed development principles and a selection of feature modeling methods.

<table>
<thead>
<tr>
<th>Principle of language parsimony</th>
<th>FORM</th>
<th>ESAPS</th>
<th>PLUSS</th>
<th>PLA</th>
<th>ODM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Principle of strict typing</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Principle of subsumption</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Principle of vertical domains</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Principle of dependency parsimony</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Principle of information parsimony</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Principle of artifact agreement</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Principle of shallow trees</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
</tbody>
</table>

**ESAPS**

It is worth reminding the reader that the ESAPS projects resulted in a selection of variability modeling practices and not a single coherent methodology. ESAPS feature models make use of feature areas to divide the top level features into independent feature sets, contributing to reduction of dependencies. The use of horizontal, rather than vertical domains encourage shallow feature trees. There are some feature attributes that are prescribed in order to perform valuations of feature model configurations. UML is regularly used in order to allow extensions of semantic constructs for feature models, but the initial set of modeling elements is reasonably limited.

**PLUSS**

The variability aspects of the PLUSS methodology are handled through a quite limited feature modeling notation, with few constructs. As the intention of PLUSS is to model the variability of use cases, the notation supports the hierarchy of use case specializations, and the structure of the feature model therefore reflect the use cases. This means that there is a natural correspondence to requirements artifacts for the features, there is little supplemental information required, and the amount of features is dependent on the use case models.

**PLA**

The feature model produced using PLA is based on the FODA-notation and does not make use of horizontal or vertical domains. It is comparatively restric-
6.7. Integration and operation

tive in terms of language features, and use the subsumption tree hierarchy of FODA. The PLA is a methodology for producing software product lines, and the feature model is one of several different models used in the analysis, modeling, and development of the product line. The feature model interacts through derivation activities with use case models and object models. There are thus strong relations to other artifacts.

ODM

Features in ODM are “significant differentiating capabilities across domain systems” [105]. Since features in ODM embody the variability of concepts, the connection to other artifacts is inherent. ODM does not prescribe details for a representation language of concepts and features, aside from mentioning specialization and generalization. The concept models are encouraged to be separated into multiple, modular models if possible. The elements of the conceptual models in ODM can have complex interactions and relations, not supported by a shallow tree structure.

6.7.2 The quality model in the development process

Recall Figure 2.3, p. 17, describing a systems and domain engineering work process with the activities prescribed by [36]. The quality model for feature models proposed in this thesis has two principal activities – prioritization and evaluation – that primarily affect the domain analysis processes in the domain engineering workflow and can be trivially integrated with the activities. With prioritization of important model qualities carried out in advance of the domain analysis, and evaluation carried out on the finished domain analysis results, the quality model can provide value to the subsequent steps of the domain engineering, in the form of increased suitability and usefulness of the models produced. The activities related to deriving development principles can be performed in parallel with the domain analysis modeling activities, while the principles themselves affect the documentation of the results from domain analysis that take the form of a feature model.

One desirable property of the quality model is that it should be applicable not only for large organizations and systems with elaborate domain engineering practices in place, but also be applicable and provide some benefits for smaller organizations. We have previously related how SMEs and small development units with expanding variability can use feature modeling in order to take control of variability related problems in the product-, problem-, and solution space.

Such that the quality model is represented today, it is accessible for small organizations, as evident by the fact that it was tested and refined using the sce-
nario just described. Because of the many variations of development processes that exist in SMEs, and the adaptability that is often associated with SMEs, it is not meaningful to prescribe one particular place in a general development process where the model fits. Examples of when a feature analysis can arise as a beneficial and useful activity are:

- a number of new product variants are introduced based on an existing product,
- the variability of existing products and components cannot be managed,
- a number of different products are unified in a common architecture,
- a technology change is made in existing products,
- a new product line is designed from scratch.

Generally speaking, the interest for performing feature modeling in a smaller organization mostly arises as a post-hoc construction, once the existing structures for managing variability are found insufficient. The analysis and modeling is then carried out as a one-off refactoring effort or re-positioning project. A generic, adaptable procedure for performing this kind of project, and utilizing the potential of the quality model in the process, is found in the technical report by the author [116].

The quality model can thus with little effort be included in most feature modeling efforts, big or small.

6.8 Chapter summary

In this chapter the second iteration of the quality model for feature models is described. The refined version of the quality model is based on the lessons learned in the case study described in the previous chapter. The refined version has a different process and proposes six quality factors. A procedure is described that let stakeholders of the feature model select and rank the quality factors that are most important for them, as well as evaluating the quality of the resulting feature model. Indicators that are found during the evaluation are turned into development principles. Development principles can be used to guide the development of a feature model in a direction where specific quality factors are promoted. Finally, an evaluation of the compatibility between the development principles and existing approaches to feature modeling is described, along with reflections on the possibility to integrate the quality model in existing development processes.
This chapter revisits the research questions, goals, requirements and intentions that the thesis intends to answer and cover. It reflects on the results of the research efforts and how the results could have been improved. Further issues for continued work are also identified.

7.1 Research results and improvements

The main results of this thesis can be summarized in a number of contributions, both from a scientific and practical perspective. The list of contributions found in section 1.4 is reproduced here with comments on how the contributions came about, and were in the thesis the contributions are described.

- A model of qualities, indicators, and dependencies for feature models.
  A quality model was designed based on previous work in quality models for software artifacts and quality of software models. The initial version (section 5.1) was revised (section 6.1) based on tests performed in a case study (section 5.2).

- An approach for finding the desirable qualities and properties of a feature model.
  The approach proposed for finding the priorities of the feature model qualities (section 6.2) is based on paired comparisons (section 5.2.1), an
7. Discussion

intuitive and straightforward approach that has been tried and tested in many other applications.

• An approach for evaluating the quality of existing feature models.

The approach for evaluating the quality of the resulting models (section 6.3) is based on the same approach of paired comparison. The approach includes finding the indicators for the specific quality factors and turning them into development principles to promote certain quality factors.

• An investigation into the industrial needs for variability management and modeling.

The industrial needs and the potential for variability modeling were investigated during a case study in a research project (section 4.3) and through a survey of software developing small and medium-sized enterprises (section 4.2). The findings have contributed to the development of the quality model.

• An evaluation of how existing approaches for development of feature models take into regard the quality and properties of the feature model.

Existing methods and approaches to feature modeling were reviewed with regard to their consideration of model quality (section 4.1). The practices found are part of the motivation for developing the approach to model quality in this thesis.

• An instrumentation and practice for finding the desired and actual qualities needed for a feature model.

Because of the simplicity and approachability of paired comparisons, the procedures (section 6.7) and instrumentation (section 6.2.3) for incorporating the model are very easy to use and require little effort to put into practice.

• An initial set of development principles for designing feature models that exhibit particular qualities.

The development principles found and formulated (section 6.6) through the validation case study provide examples and a starting point for describing practices and guidelines that promote certain quality factors in a feature model.
7.1. Research results and improvements

- Empirically grounded considerations to equip existing methods for feature model development with.

  The integration (section 6.7) of this approach to feature model quality with existing feature modeling approaches, combines established knowledge and experience regarding feature modeling, with the new concepts for higher quality in the feature models.

- FMWIKI, a web-based feature modeling tool used during the case study.

  The tool FMWIKI (section 5.2.7) reached a fully functional state and level of maturity were it contributed to the modeling efforts in the validating case study.

We can reflect on the research results and contributions using a few questions that put the finger on delicate spots when it comes to scientific research, which will be done in the following sections.

7.1.1 Have the research questions been answered?

We can summarize what we have learned through the work described in this thesis by answering the research questions posed in the first chapter.

- Do properties of a feature model affect its usefulness in different contexts?

  We have seen that the stakeholders of a feature model can require different qualities from the model in order to find it suitable for their respective needs. We have learned that properties of a feature model do affect how the quality of the feature model is perceived by its users. We have not conclusively found how the context, as described in the resulting method, affect the requirements of the feature model. We have seen that the usage scenario of a feature model can function as an important input to the modeling activity.

- How can a feature model’s properties be evaluated with regard to their effect on the quality of the feature model?

  We have learned that the quality factors that are present and prominent in a feature model can be practically evaluated using simple paired comparisons. We have seen that the definitions of a number of quality factors for feature models can be used to reason about the emergent quality of a feature model and that properties that promote those quality factors can be identified by the stakeholders of a model.
7. Discussion

- What properties of a feature model can be changed or modified to influence the quality of the feature model?

We have learned that based on an evaluation of prominent quality factors in a feature model, indicators of those quality factors can be identified. We have seen that many of the indicators are of a form and type that can be modified during the development of the feature models. Such indicators can be used as input for deriving development principles that promote quality factors in the feature models.

In summary, we learned that the quality of a feature model matters to its stakeholders and that the development of a feature model can be guided by development principles that make the model better suited for its purpose.

7.1.2 Could we have learned the same with less effort?

The question of whether we could have learned the same with less effort is a question of how well the time spent on this research was invested. The early stages of the research was carried out as literature surveys and working in the initial case study to see if there was a need for investigating quality of feature models. The initial case study did not provide a very rich material aside from some questions to be looked at. Had the first case study provided empirical material that could have been used to evaluate the approach to feature model quality, the need for additional case studies could have been mitigated. On the other hand, the second case study provided a very suitable setting to test the approach in, which would not have been matched by the first case.

The survey that was conducted provided unique material that, to the best of the author's knowledge, is not available from any previous work. From an administrative point of view, surveys are arduous and cumbersome to work with. However, since the results of the survey were very interesting, and also contributed to other research questions than the ones mentioned in this thesis, the effort was well spent.

The experimental approach described in section 3.5.1 did not pan out well, which is indeed unfortunate. Although the author takes a skeptic's view on the value of software engineering experiments on students, a series of replicated experiments concerned with usability of feature models could have provided interesting ideas and input. The convenience and accessibility of such an approach was however outweighed by the lack of validity evident in the pilot experiment.

The answer to this question is thus that we could not have gotten the same results by any changes in the methodology or planning of the work.
7.1.3 What could we have learned in addition?

Given that we could not have learned the same with less effort, could we have learned more from the same effort? In retrospect there are indeed some parts of the work that could have given additional results if there had been some changes in the execution of the activities.

The work in the initial case study could not be controlled to any significant extent, but the survey was wholly within control and could have contributed additional material. The survey that was distributed contained many questions that were not directly related to the issues treated in this thesis. The answers to many of those questions could be analyzed and related to the questions that do concern the topics in this thesis, in order to see if more correlations can be detected. However, the survey’s format and structure was not optimal for extensive analysis of that kind, which was a set-back detected only when the analysis had begun. There were not much room for additional questions to be included in the survey. A balance had to be struck between the questions that concerned variability and the questions that concerned other aspects of software development.

The second case study provided fully adequate material for the validation purposes and met the expectations from the scientist’s point of view. Having said that, the case study could have contributed more material for the practitioner, in the form of more indicators and additional tested guidelines. Extracting that from the case study could have been possible with some changes in the work organization. As it were, the indicators were found by the evaluators of the resulting feature model, without intervention of the modeling leaders. Had the evaluators been “coached”, there may have been more indicators mentioned. This could on the other hand have led to the evaluators only finding the indicators that the modeling leaders wanted them to find. Again, a balance had to be struck.

7.1.4 Are the results scientifically grounded?

In chapter 3 the research methodology for this thesis is described. As stated there, the attitude to different research approaches has been that they can complement each other, rather than compete against one another. Having said that, are the results of the work sufficiently grounded from a scientific perspective, or are they a gallimaufry put together by things that just happened to be available?

Part of the answer lies in how the results have been validated. In engineering sciences, it is seldom practical to attempt validation through falsifiable statements, since the number of variables involved are more than enough to find exceptions to any rule. Instead, we need to apply a stepwise refinement of the-
ories and claims, taking new findings into regard as the results are applied in engineering settings. This approach is reflected in the revision of the initial quality model that was proposed, as well as in the feedback mechanisms present in the presented approach to ensure quality in feature models.

Another part of the answer is in the procedures and processes followed throughout the work. Recall from section 2.3 that quality in the process is seen as a means to ensure quality in the product. The work has followed sound practices with documentation and transparency along the way. All significant, and some insignificant, results of the work have been published and subjected to peer-review, ensuring that the results have held a certain level of merit.

7.1.5 Did we pass the test of reality?

As mentioned in section 3.6, it is seen as important that the results of this research can contribute to the engineering practice, if not immediately, then at least in the long run. Several of the contributions are presented as material that can serve a practitioner in producing feature models that are suitable for their intended purposes. While the initial quality model was based largely on theoretical studies and adaptations of best practices and experiences from other research into quality, the refined model is based on the findings from the second case study.

Assuming that the case study is an appropriate description of actual circumstances where the quality model would be applied, it would follow that the research did pass the test of reality. Given what was found from the survey concerning the variability management situation in SMEs, the case study is representative for SMEs and a suitable context were the quality model can be used. The survey and the case study only have local validity and there are bound to be many situations were the model might not hold. But for the test that the approach was put to, it did pass.

7.2 Requirements revisited

In section 4.4, a list of high level requirements on an approach to increase feature model quality was presented. Table 7.1 shows the extent of requirements fulfillment that the refined approach reaches. In summary, the approach fulfills all the requirements but one. Neither the survey nor the case studies provided enough material concerning the effects of business domain on the variability management. The initial case study concerned embedded systems in the automotive domain, the survey received answers from many different domains, and the second case study was in the mobile applications domain, but at the end of
7.3 Future work

The research presented in this thesis has answered the original research questions posed, but obviously there are many other questions that can be looked at next.

As mentioned earlier in this chapter, the research could have provided a wider array of indicators and development principles. It is definitely necessary to evaluate more feature models and find more indicators for the quality factors. The indicators also need to be analyzed in order to derive development principles. This work would not be so much a matter of research, as it would be expansion and fine-tuning of the proposed approach. It could be argued that the definition of specific indicators and principles is the task of the individual organization applying the quality model, but if the indicators are kept undisclosed, the wheel will be invented over and over. It would therefore be useful to construct a method supporting the identification and dissemination of indicators and principles in the community using the approach.

Experimentation has already been mentioned several times. Designing and executing experiments that gather supporting or contradicting evidence for indicators and principles could provide some level of validation. A problem of experimenting with the development principles is that there is considerable effort necessary to develop a feature model of a size that matches an industrial model. This means that testing all the indicators and principles in isolation would be very unpractical. The challenge of finding an appropriate experiment design is indeed interesting.

Again, the effect of business domain is not clear and requires more research. The only feasible way to investigate this is to apply the model in different industrial domains and carefully select case studies that allow a characterization of the influencing properties. This is not attainable through experimentation, surveying or other distributed effort and is therefore utmost challenging.

When it comes to quality, more is better, and it can be difficult to choose between two quality factors. An interesting question is if it might not be necessary in the first place. Perhaps it is possible to create a perfect feature model where all quality factors are fully present and there are no trade-offs needed. Part of the answer to this question lies in quantifying the preferences and evaluations. At the moment, the prioritization states an order of preference regarding the quality factors, but the exact ratios between the factors are not sufficiently explored. It is thus not known if it is possible to have a certain level of one quality factor,
Table 7.1: Fulfillment of requirements for a quality model for feature models.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>The approach should provide an empirically founded model for how qualities in feature models are related to one another.</td>
<td>There are initial results on this aspect as presented in Table 6.1.</td>
</tr>
<tr>
<td>The approach should consider the stakeholders’ intended use for the feature model.</td>
<td>This is inherent to the approach used.</td>
</tr>
<tr>
<td>The approach should provide the modelers with guidelines for constructing feature models that are suited for their intended purpose.</td>
<td>The development principles fills this function.</td>
</tr>
<tr>
<td>The approach should provide guidelines and a set of indicators for evaluating the prominence of the qualities in the model.</td>
<td>This is catered to by the approach.</td>
</tr>
<tr>
<td>The approach should be possible to integrate with existing methodologies for feature model development.</td>
<td>The proposed approach can be integrated with little effort.</td>
</tr>
<tr>
<td>The approach should be palatable for small organizations.</td>
<td>The approach is simple and fast to use, and can be applied in common scenarios where an SME would consider feature models.</td>
</tr>
<tr>
<td>The approach should take into regard the domain of the model and the requirements posed on the model because of the domain.</td>
<td>This is to a great extent missing from the model.</td>
</tr>
</tbody>
</table>
given a level of another. In other words, is it possible to have more than $x$ for one quality factor, if you also want $y$ for another or do the two factor only have $x + y$ to share between themselves?

Finally, the quality attributes of the initial quality model have not been employed in the prioritizations and evaluations to the extent that the quality factors have. As stated previously, the number of paired comparisons between all the quality attributes are too many to be practical. The suggested approach is instead to only look at the detailed attributes once the factors have been sorted out. In order to do so, more work is required to establish and validate the influences of the quality attributes on the quality factors. Table 7.2 shows the current hypothesis regarding the relations between the quality attributes and the quality factors. The influences are derived from the same sources as the quality factors and attributes, e.g. [76], but are adapted for applicability to feature models. Validating this hypothesis could be a useful contribution to the quality model.
### Table 7.2: Hypothesized contributions of quality attributes to quality factors.

<table>
<thead>
<tr>
<th>Quality Attribute</th>
<th>Changeability</th>
<th>Reusability</th>
<th>Formalness</th>
<th>Veracity</th>
<th>Mobility</th>
<th>Usability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adaptability</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Extensibility</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Stability</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Modularity</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Self-containedness</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Analyzability</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Conformance</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Consistency</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Testability</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Accuracy</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Redundancy</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Completeness</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Reliability</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Robustness</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Installability</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Interoperability</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Portability</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Complexity</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Learnability</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Structuredness</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Understandability</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Acceptability</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Accessibility</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Communicativeness</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Visibility</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>
Chapter 8

Conclusion

Having discussed the contributions of the thesis, and retrospectively evaluated the efforts that led up to those contributions, this final chapter summarizes the conclusions drawn from the work and the merits of those conclusions.

The research questions

- Do properties of a feature model affect its usefulness in different contexts?
- How can a feature model’s properties be evaluated with regard to their effect on the quality of the feature model?
- What properties of a feature model can be changed or modified to influence the quality of the feature model?

have been answered. We can conclude that properties and traits of a feature model affect how stakeholders see the quality of a feature model, and that indicators of quality factors can be found and influenced in order to impact the usefulness of the feature model. This thesis proposes approaches to find, evaluate, and manage such properties, and provides a “starter set” of indicators based on empirical studies.

The underlying reason for trying to answer the posed research questions, is the notion that feature models are generally not reaching their full potential and that systematic efforts can be taken to make feature models more useful. While
8. Conclusion

feature models have been the subject of many changes to notations, new metamodels and extensions of supposed expressibility, the focus has almost always been to raise the quality of the products that are described by the feature model, rather than the feature model itself. But in order to draw on the benefits of modeling the products, the users of the model must be considered first.

The intellectual merit of the contributions in this thesis lies in this work being the first to systematically consider the users of the feature model, rather than the users of the artifacts described by the model. The investigation of variability management practices and the perceived potential for SMEs is also of merit for future research. It offers insight into the state of an important segment of software developing organizations that is often neglected in software engineering research.

The results of the work propose a practical means to improve the chances of producing a feature model that will be put to good use. The results hold engineering merits in having not only a solid basis in established theory and prior engineering practice, but also having been validated through application in practice. Putting the contributions of this thesis into use is easy, and can provide immediate benefits with a small investment of effort. Care has been taken to make the approach accessible and straight-forward.

In summary, the work in this thesis caters to both science and practice. It is a starting point for exploring the complex, and up until now unchartered, interactions of quality factors and properties of feature models, that will increase the understanding of how feature models contribute to producing software-intensive systems. It is also an approachable addition to the feature modeling process, that will benefit the practice of variability modeling.


8. Bibliography


8. Bibliography

In *Proceedings of the 5th Object-Oriented and Internet-Based Technologies*, 2004. [Cited on p. 1, 29]


List of Figures

1.1 An example of a partial feature model describing a car. . . . . . . 4
2.1 The relation between original and model according to Stachowiak 13
2.2 The relationship between descriptions of the world and the ma-
chine, forming the model. [55]. . . . . . . . . . . . . . . . . . . . 13
2.3 Overview to Systems and Domain Engineering. [36] . . . . . . . . 17
2.4 The equivalent constructions of mutex and requires relations . . 28
2.5 The FODA-notation of feature diagrams. [66] . . . . . . . . . . . 30
2.6 The FORM-notation of feature diagrams. [70] . . . . . . . . . . . 31
2.7 The Eisenecker-notation of feature diagrams. [21] . . . . . . . . . 32
2.8 The Riebisch-notation of feature diagrams. [99] . . . . . . . . . . 33
2.9 Quality in the life-cycle. [54] . . . . . . . . . . . . . . . . . . . . . 37
3.1 Illustration of thesis work ordered by time and phase. . . . . . . 58
3.2 Illustration of how research activities, results and publications
are related. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 59
4.1 An example of PLUSS, showing the relations between features,
use cases and scenarios . . . . . . . . . . . . . . . . . . . . . . . . . 65
4.2 Example of simplification of feature model structure. [70] . . . . 70
4.3 Example of survey question and alternatives. . . . . . . . . . . . 76
4.4 Number of employees in the respondents’ organizations. . . . . . 78
4.5 Use of software in the respondents’ products. . . . . . . . . . . . 79
List of figures

4.6 Organization roles of respondents. 79
4.7 Documentation of product structure used by respondents. 81
4.8 Problems due to lack of product structure documentation. 83
4.9 Principles used for problem descriptions. 83
4.10 Ratio of problems that are essentially similar between different products. 84
4.11 Ratio of products with a common base and individual adaptations. 84
4.12 Perceived shortcomings due to lacking problem structures. 85
4.13 Potential improvements with a well-designed problem space. 86
4.14 Ratio of solution similarity among different products. 87
4.15 Solution reuse mechanisms used by the respondents. 87
4.16 Satisfaction level of structures for products, problems and solutions. 88
4.17 Potential improvements with a well-designed solution space. 88
4.18 Significant factors for making changes in solution space variability structures. 89
4.19 Comparative means between small and medium/large organizations. 94
4.20 The growth of electric/electronic sub-systems in cars. [120]. 97
4.21 The SEMCO feature model in graph format. 98
4.22 The SEMCO family models of the hardware and runtime systems. 100

5.1 Overview of the initial quality model components. 107
5.2 Intuitive mapping of model elements in ISO 9126, the initial proposal for feature model quality, and the quality model by Boehm. 109
5.3 Comparison matrix, filled comparison matrix and ranking using paired comparison. 117
5.4 The Tactel business drivers as illustrated by Tactel. [111]. 119
5.5 Overview of the communication flows in the framework for mobile application development. 119
5.6 Radar graph comparing the qualities that the stakeholders prioritize in a feature model, and the evaluation of the resulting feature model. 128
5.7 Example of a preference graph with circular triads. [23]. 140

6.1 Overview of the components in the refined quality model. 146
6.2 A practical meta-model for feature models. 148
6.3 Sample of form used for prioritization of quality factors. 153
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Quality model for external and internal quality.</td>
<td>38</td>
</tr>
<tr>
<td>2.2</td>
<td>Quality model for quality in use.</td>
<td>39</td>
</tr>
<tr>
<td>2.3</td>
<td>List of qualities in models suggested from literature.</td>
<td>40</td>
</tr>
<tr>
<td>2.4</td>
<td>List of metrics for qualities suggested by various quality models.</td>
<td>41</td>
</tr>
<tr>
<td>3.1</td>
<td>Research methods and their applicability. [128]</td>
<td>49</td>
</tr>
<tr>
<td>3.2</td>
<td>Research methods and their validity types. [127, 40, 128]</td>
<td>55</td>
</tr>
<tr>
<td>3.3</td>
<td>Objections to empiricism. [28]</td>
<td>56</td>
</tr>
<tr>
<td>4.1</td>
<td>Summary of evaluation criteria and feature modeling approaches.</td>
<td>73</td>
</tr>
<tr>
<td>4.2</td>
<td>Number of respondents active in various domains.</td>
<td>80</td>
</tr>
<tr>
<td>4.3</td>
<td>Amount of product customization relative to the types of product descriptions used in the organization, numbers indicating the number of respondents.</td>
<td>81</td>
</tr>
<tr>
<td>5.1</td>
<td>Quality factors and attributes</td>
<td>111</td>
</tr>
<tr>
<td>5.2</td>
<td>The average number of tokens given each quality factor during prioritization. Higher number indicates higher priority.</td>
<td>125</td>
</tr>
<tr>
<td>5.3</td>
<td>Some properties of the resulting model.</td>
<td>126</td>
</tr>
<tr>
<td>5.4</td>
<td>The average number of tokens given each quality factor during Evaluation. Higher number indicates higher presence of quality factor in the feature model.</td>
<td>127</td>
</tr>
</tbody>
</table>
List of tables

5.5 Examples of specific indicators found in the case study and their influence on the quality factors. ........................................ 130
5.6 Descriptions of indicators for supported model elements. ..... 132
5.7 Descriptions of indicators for placement of features in hierarchies. 134
5.8 Descriptions of indicators for structure and amount of relationships and dependencies. .......................... 135
5.9 Descriptions of indicators for amount of supplemental information. .................................................. 136
5.10 Descriptions of indicators for overview and correspondence to actual artifacts. ................................. 137
5.11 One participant’s choice matrix for the prioritization of quality factors. .................................................. 141
5.12 A double sorted choice matrix. ........................................ 141
5.13 Examples of specific indicators found in the case study and their values. ............................................... 143

6.1 Relations between quality factors. .............................. 150
6.2 The influence of development principles on the quality factors. 158
6.3 The compatibility between proposed development principles and a selection of feature modeling methods. .............. 162

7.1 Fulfillment of requirements for a quality model for feature models. 172
7.2 Hypothesized contributions of quality attributes to quality factors. 174