

Deficiencies in Feature Models

Thomas von der Maßen, Horst Lichter

{vdmass, lichtner}@cs.rwth-aachen.de
Research Group Software Construction
RWTH Aachen
Ahornstr. 55, 52074 Aachen

Abstract. Software product lines are characterized through common and variable parts. Modeling variability is one of the most important tasks during the analysis phase. Domain analysis and requirements elicitation bring up a huge amount of requirements and dependencies between product characteristics. Feature modeling is one approach to deal with complexity in expressing several requirements in features and structure them hierarchically in feature diagrams. Unfortunately these feature models become very complex as well. Therefore it is necessary to develop and maintain feature models very carefully with respect to redundancy and consistency. As feature models are not only used for domain modeling, but for product derivation in product line development as well, inconsistent feature models will limit the chance to build consistent product configurations. Hence, it must be defined, what is meant by consistency and redundancy in the context of feature models. Experiences show that an adequate tool support is needed to manage the feature models and to support automatic detection of redundancies and inconsistent models and product derivations. Our research group has developed a prototype of a requirements engineering tool that supports feature modeling and provides automatic consistency checks.

1. Introduction

The development of software-intensive systems shifts more and more from single product to software product line development. The development of a software product line requires a comprehensive understanding of the domains, targeted by the product line. A detailed and thorough analysis of the domains must be performed and the results have to be documented in a coherent way. The research area of domain analysis deals with the investigation and modeling of domains. As van Deursen and Klint pointed out in [9], "domain analysis originates from software reuse research, and can be used when constructing domain-specific reusable libraries, frameworks, languages, or product lines". A number of analysis methodologies exists, of which Organization Domain Modeling (ODM [10]), Feature-Oriented Domain Analysis (FODA [1]), and Domain-Specific Software Architectures (DSSA [11]) are the most popular ones.

Furthermore FODA is one approach to model high level reusable characteristics in terms of *features*. Kang et al. define a feature as "a prominent user-visible aspect, quality, or characteristic of a software system or systems" [1]. Since its introduction

in 1990, feature modeling has been more and more appreciated by requirements engineers and domain analysts. There are several approaches for modeling features that are all based on the initial approach presented by Kang et al. (e.g. [2], [3], [4], [5], [6]). Goals and purposes of feature modeling shifted from domain modeling to product line modeling and feature models are used in many different application areas. Additional modeling elements, groupings and views have been introduced to fulfill the requirements that were put on feature modeling. Whereas Lichter et al. provide a good survey of the different feature modeling approaches [8], the various application areas in which feature modeling were applied are documented by Müller et al. [12]. That work presents, that feature modeling has been successfully applied in portfolio and project management, architecture derivation and mapping, commissioning, and of course in domain analysis and product derivation in the context of product line development.

In this paper we focus on using features for domain modeling and product derivation. Independently from the application area, feature models and the corresponding diagrams become very complex and difficult to survey and maintain, unfortunately. Therefore the feature models must be of high quality. The quality of a feature model is determined by how adequately it captures a given domain and by the integrity of the model itself with respect to the used modeling elements. As the adequate capture of the domain can only hardly be analyzed and reviewed by domain experts, the integrity of the model can be determined by the occurrence of redundancies, anomalies and inconsistencies. We see the following correlations:

- Maintainability of the model is very much influenced by redundancy
- Readability may be increased if important information is modeled redundantly. On the other hand, redundantly modeled information is very hard to maintain and side effects, while changing the model, are very likely to occur.
- The stability of a model depends mainly on anomalies and consistency. Inconsistent models contain contradictory information. If the domain model is inconsistent it cannot be used for product derivations, because the derived products may have inconsistent configurations, as well.

Hence, there is a strong demand for consistent domain and derived product models. An adequate tool support is needed to manage features, their relationships and dependencies to guarantee the development of consistent models. Therefore the detection of redundancies and inconsistencies is a very important requirement on feature modeling tools. The strong demands for a tool supporting software product line development have been described in detail in [7]. In order that tools can support feature model developer, it must be defined what is meant by redundancy and consistency in feature models. Guidelines for detecting redundancies and inconsistencies are necessary and guidelines for resolving these circumstances are desirable, as well.

In this paper we focus on redundancy, anomalies and inconsistency in feature models. We define these properties and list typical occurrences in feature models. We have organized this paper as follows: In chapter 2 we briefly present existing work in this research area, which results have been produced and how these results can be used for

our work. In chapter 3 we define redundancy and inconsistency in the context of feature models and give an enumeration of different types. Furthermore we give some guidelines to resolve these deficiencies. In chapter 4 we show how the detection of redundancies and inconsistencies in feature models can be supported by a tool. Finally, in chapter 5 we give a brief outlook on our future work.

2. Related Work

Even though, feature modeling became very popular in the last few years, only a few researches have been engaged in consistency topics. Most research results deal with formalization of feature models, which is necessary for tool support and normalization of feature models. Furthermore, guidelines are given for developing feature models.

Van Deursen and Klint propose in their work [9] a Feature Description Language (FDL) to describe features in a textual language and to support domain-specific language design. Based on their textual representation of feature diagrams they define rules to manipulate these models with respect to normalization and expansion to a disjunctive normal form. The following rules have been defined:

- Normalization rules, to simplify the model by eliminating duplicate features and degenerate cases of various constructs
- Variability rules, to determine the cardinality of possible feature combinations, based on the modeled variability
- Expansion rules, to expand a normalized feature expression into a disjunctive normal form
- Satisfaction rules, to determine which disjuncts satisfy given constraints, based on the disjunctive normal form

The work gives a good basis for further analysis of feature models. Unfortunately, the coherence of domain relationships, variability and dependencies is not discussed, which we see as one of the main sources of degenerated and inconsistent feature models.

Czarnecki and Eisenecker deal with normalization of feature diagrams, too [3]. Normalization is achieved by transforming combinations of child features with different types of variability to child features with a single type of variability. The combination of various types of variability includes for example that child features of an alternative-relationship are modeled as optional features. The combination of various types of variability in a single parent-child relationship should be avoided, to eliminate any possibility of misinterpretations and is not allowed in all feature modeling approaches.

Furthermore, Dörr gives a quite good overview of guidelines for inspecting domain model relationships [14]. This work classifies non-domain specific relationships and

gives guidelines how to make them explicit and how to use them to improve the quality of the domain model. Again, the coherence of domain relationships, variability and dependencies is not discussed and there is no discussion about modeling features and relationships redundantly.





3. Redundancy and Consistency in Feature Models

In this section we describe what is meant by redundancy, anomalies and inconsistency and how these deficiencies can occur in feature models with respect to the used modeling elements, most feature modeling approaches offer.

3.1 Feature Modeling Concepts

As pointed out in the introduction there are many feature modeling approaches which most of them take the FODA modeling concepts as a basis. Table 1 summarizes the relationships, most approaches offer. It must be explicitly mentioned that FODA characterizes features to be either mandatory or optional. With respect to feature reuse and the fact that features can be mandatory in one context (domain) and optional in another, variability should be expressed through a relationship between two features. Therefore we model mandatory and optional relationships with the semantic described in Table 1.

Table 1: Feature modeling relationships

Relationship	Type	Semantic	Characteristic	Notation
Domain Relationship	Mandatory	If the father feature is selected, the child feature must be selected as well		 <p>Father Child</p>
	Option	If the father feature is selected, the child feature can but need not to be selected		 <p>Father Child</p>
	Alternative	If the father feature is selected, exactly one feature of the alternative-child-features must be selected	Implicit mutually exclusion between alternative-child-features	 <p>Father Child 1 Child 2</p>
	Or	If the father feature is selected, at least one feature of the or-child-features must be selected		 <p>Father Child 1 Child 2</p>

Relationship	Type	Semantic	Characteristic	Notation
Dependency	Implication	If one feature is selected the implied feature has to be selected as well, ignoring their position in the feature tree	Transitive	\longrightarrow
	Exclusion	Indicates that both features cannot be selected in one product configuration and are therefore mutually exclusive	Symmetric	\longleftrightarrow

Hence we define mandatory features as follows:

mandatory feature

a feature is mandatory, if it is connected to its father feature through a mandatory relationship.

Nevertheless we follow the definition of a feature model given by the FODA approach [1]. That means a feature model is based on a tree structure, where the root represents the concept node. The constraint that the domain relationships build a tree structure and not a graph structure, is motivated thereby that most of the features modeling approaches propose a tree structure and that tree structures are much more easier to understand and to communicate to stakeholders in the domain modeling an product configuration processes. Taking the feature model as a basis for product configuration, the selection process of which feature should belong to a special product and which not, starts at the concept node and follows the tree down to the leafs. A child feature can only be selected if the father feature has been selected as well. At a variation point – that means a feature which is connected to child feature(s) by an optional, alternative or or-relationship – the product configurator has the choice which child feature(s) to select. We define a selectable feature as follows:

selectable feature

a feature is a selectable feature, if it is a child feature of an optional, alternative or or-relationship

Though the core set of relationships is very small, it misleads the modeler to build redundant and inconsistent models, as well as models which contain other types of anomalies. The aim is to build redundancy-free, anomaly-free and consistent feature models to guarantee modifiability - like maintainability, scalability and extendibility - and reusability.

3.2 Redundancy, Anomalies and Inconsistency

In this subsection we want to characterize several undesirable properties of feature models. To determine the quality of a feature model we want to analyze whether the model contains these specific characteristics which might point to a falsely modeled domain. These characteristics are categorized by the severity of the problems that might arise from them. The categories are:

- Redundancy – A feature model contains redundancy, if at least one semantic information is modeled in a multiple way. If information is modeled redundantly, the maintainability of the model may decrease. On the other hand it must be explicitly mentioned, that redundancy is not always a bad thing. If redundancy helps to increase readability and understandability of the model, it is an adequate means that has to be applied very thoroughly. Here the model developer has to make a tradeoff between the quality criteria maintainability and readability. Therefore redundancy is regarded as a light issue.
- Anomalies – A feature model contains anomalies, if potential configurations are being lost, though these configurations should be possible. The reason for this circumstance is that some senseless information has been modeled. If a feature model shows the characteristic of anomalies, it likely captures the domain wrongly. Anomalies in a feature model are regarded as a medium issue.
- Inconsistency – A feature model contains inconsistencies, if the model includes contradictory information. Therefore, inconsistent models contain information that is conflicting with some other information in the same model. Inconsistency leads to the fact that in most cases no consistent product configuration can be derived from the domain model. Thus, inconsistencies are characterized as a severe issue.

The problem of modeling redundancies, anomalies and inconsistencies arises - among other issues - because of the overlapping semantic of domain relationships and dependencies. For example: The alternative-relationship implies that all alternative-child features are mutually exclusive. In the following subsections we analyze representatives for redundancy, anomalies and inconsistency, a feature model might have.

For the descriptions of the different occurrences of redundancies, anomalies and inconsistencies, we define two new concepts:

full-mandatory feature

a full-mandatory feature is a mandatory feature, whose predecessors in the feature tree are all mandatory.

relative-full-mandatory feature

a feature F2 is a relative-full-mandatory-feature to a feature F1 if F2 is a mandatory feature and all features on the path to its predecessor F1 are all mandatory. F1 itself has not to be a mandatory feature, stringently.

3.3 Redundancy

We have identified two different sources, leading to redundant information in feature models. The trivial case, modeling multiple domain relationships or dependencies of the same type between features, will be ignored at this point.

Redundant modeled features

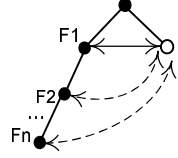
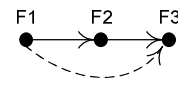
A feature is modeled redundantly if it appears multiple times in the feature model. That means all occurrences of the feature are connected through domain relations to different father features. This is often the case when the model is constrained to be in a tree structure. Thus, a feature is not allowed to have more than one parent feature and therefore it appears at least two times within the feature tree. In this case it must be decided whether the redundancy has been explicitly modeled that means the multiple appearance of the feature is desired, or if the redundant feature has been accidentally modeled. This might be the case, when the feature is connected through different domain relationships, for example, it is part of a mandatory relation and an optional relation.

Combinations of domain relationships and dependencies

Some kinds of combinations between domain and dependency relationships lead to redundant information. Table 2 lists those combinations. Typically, the redundancy can be resolved by eliminating the superfluous dependency. In the examples, the superfluous relationship is visualized by a dashed line.

Table 2: Redundant modeled relationships

Combination	Description	Example
Mandatory and Implication	<p>A full-mandatory feature is implied by another feature. As the feature is already full-mandatory, the implication is superfluous.</p> <p>Another case exists if a feature F1 implies a feature F2 and F2 is relative-full-mandatory to F1.</p>	
Alternative and Exclusion	A mutual exclusion is modeled between alternative-child features. As the alternative relation implies a mutual exclusion between the child features the dependency is superfluous.	
Multiple Implications	A feature is implied by multiple features F1,...,Fn whereas F1 is a parent of F2,...,Fn and F2,...,Fn are relative-full-mandatory to F1. The implications from F2,...,Fn are superfluous.	

Combination	Description	Example
Multiple Exclusions	A feature is mutual exclusive to multiple features F_1, \dots, F_n whereas F_1 is a parent of F_2, \dots, F_n and F_2, \dots, F_n are relative-full-mandatory to F_1 or imply each other. The implications from F_2, \dots, F_n are superfluous.	
Transitive Implications	A feature F_3 is directly implied by the features F_1 and F_2 and F_2 is implied by F_1 . As F_3 is already implied by the transitive implication from F_1 through F_2 , the direct implication from F_1 to F_3 might be superfluous.	

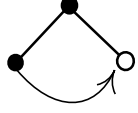
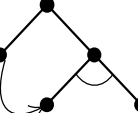
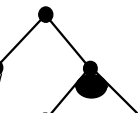
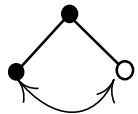
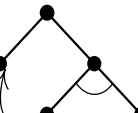
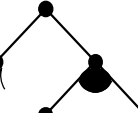
3.4 Anomalies

In this subsection we focus on feature models which contain anomalies. Such feature models indicate a wrong modeled domain. Problems arise, as the choice of possible product configurations by resolving variation points is restricted, by senseless modeled information. Again, combinations of domain and dependencies relationships are responsible for the senseless models. To be more precise, the model contains senseless information whenever a dependency is modeled between a selectable and a full-mandatory feature. As the full-mandatory feature is always part of a product configuration the selectable feature is not selectable anymore, but the selection is constrained by the dependency from the full-mandatory feature. This circumstance will lead to misinterpretations of the feature model. Table 3 lists identified types of anomalies in a feature model. Nevertheless, it must be mentioned that these cases neither provide any redundant information, nor any inconsistencies.

3.5 Inconsistency

The problems discussed in section 3.3 and 3.4 point to a possibly wrong captured domain. Though the modeled information is redundant or senseless it includes no contradicting information. In this section we discuss inconsistent feature models containing contradicting information which indicates a wrong captured domain. Whereas from the feature models which contain redundancies or anomalies, a product derivation is possible, inconsistencies make it impossible to derive consistent product configurations in most cases and have therefore to be resolved. We have identified inconsistencies on two levels. First, inconsistencies may occur at the domain level and second, they may occur at the product configuration level. In the following, we analyze these levels in more detail.

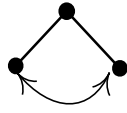
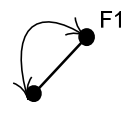
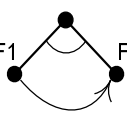
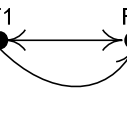
Table 3: Anomalies in feature models

Combination	Description	Example
Implication and Option	An optional feature is implied by a full-mandatory feature. Consequently the optional feature is not optional anymore but becomes a full-mandatory feature, as well.	
Implication and Alternative	An alternative-child feature is implied by a full-mandatory feature. Consequently the implied feature becomes a full-mandatory feature, too and the remaining alternative-child feature(s) can never be selected.	
Implication and Or	An or-child feature is implied by a full-mandatory feature. Consequently the or-child feature becomes a full-mandatory feature, too. Now the implied or-child feature is always part of the selection when resolving this variation point.	
Exclusion and Option	An optional feature is mutual exclusive to a full-mandatory feature. Consequently the optional feature can never be chosen when resolving this variation point.	
Exclusion and Alternative	An alternative-child feature is mutual exclusive to a full-mandatory feature. Consequently the alternative-child feature can never be chosen and always one of the remaining alternative-child features has to be selected when resolving this variation point.	
Exclusion and Or	An or-child feature is mutual exclusive to a full-mandatory feature. Consequently the or-child feature can never be chosen and that at least one of the remaining child features has to be selected when resolving this variation point.	

3.5.1 Inconsistency on the domain level

Inconsistencies on the domain level represent contradicting information in the domain feature model. Independently from resolving variation points the model includes contradictions which can never be fulfilled. We have identified four different situations which lead to inconsistent information. These are summarized in Table 4. Only in the latter two examples, a consistent product configuration can be derived and only if the feature F2 is part of the product configuration and the feature F1 is not.

Table 4: Inconsistencies on the domain level

Combination	Description	Example
Exclusion between full-mandatory features	A mutual exclusion between two full-mandatory features has been modeled. As both features have to be part of every potential product they must not exclude each other.	
Exclusion between relative-full-mandatory features.	A mutual exclusion between a feature F1 and a relative-full-mandatory feature to F1 has been modeled. If F1 is selected in a product configuration the relative-full-mandatory feature has to be selected as well and they must therefore not exclude each other.	
Implication between alternative-child features	An implication between two alternative-child features has been modeled. As the alternative-child features are mutual exclusive, one feature must not imply the other.	
Exclusion and Implication	A mutual exclusion and an implication have been modeled between two features, simultaneously. Two features cannot be mutual exclusive and implied at the same time.	

Though the mentioned inconsistencies can always be resolved by eliminating the dependency, these inconsistencies point to a falsely captured domain which normally leads to a complete restructuring of parts or even the whole feature model.

3.5.2 Inconsistency on the product configuration level

Inconsistencies on the product configuration level represent conflicting or incomplete product configurations. A product configuration can be derived from the domain feature model by resolving variation points and including all features which have to be part of the product, regarding the domain relationships and dependencies. Inconsistencies can arise, if not all features which should be in the configuration or if conflicting features are selected for a specific configuration. Table 5 lists inconsistencies that can appear in a product configuration.

These inconsistencies can be avoided by strictly obeying the semantics of the modeled domain relationships and dependencies. As this is a very difficult task in very large domain models, an adequate tool supported is needed that guides the product configurator through the steps of product derivation.

Table 5: Inconsistencies on the product configuration level

Case	Description
Missing full-mandatory features	One or more full-mandatory features have not been selected for a specific product configuration. As full-mandatory features should be part of any product configuration, the configuration is incomplete.
Missing mandatory child features	All mandatory child features of a feature have to be selected for a product configuration. Otherwise, there is an incomplete product configuration.
Wrong resolving of an alternative-variation point	Either no or more than one alternative-child feature is selected. As exactly one alternative-child feature has to be selected, conflicting features are part of the product configuration.
Wrong resolving of an or-variation-point	No or-child feature is selected. As at least one or-child feature has to be selected, the configuration is incomplete.
Missing implied features	An implied feature is not selected. As the implied feature has to be selected if the implying feature is selected, the configuration is incomplete.
Mutual excluded features	Two mutual excluded features have been selected. As mutual excluded features cannot be both part of a configuration, the configuration contains conflicting information.

3.6 Normalized feature models

We define a feature model as normalized, if it does not contain any characteristics described in section 3.3, 3.4 and 3.5. Therefore no redundancies, no anomalies and no inconsistencies are contained in a normalized feature model.

Transforming a redundant feature model into a normalized feature model is difficult, if features have been modeled redundantly. Either it comes along with a complete restructuring of the tree or the strict tree structure has to be abolished and the model has to be transformed into a feature graph. In this case, the former redundantly modeled feature has two (or more) father features. Redundantly modeled relationships, mentioned in section 3.3, can be resolved by deleting the superfluous dependency (illustrated as a dashed line in Table 2). Nevertheless, it must be mentioned that this step should not be performed automatically by a tool, but domain experts have to analyze those situations, since automated transformations might damage the intention of the domain model.

Eliminating anomalies in a feature model is difficult, too. In general there are three possibilities to transform the model containing anomalies into a normalized feature model:

- Deleting the irritating dependency. As this step influences the modeled semantic in a very serious way, it cannot be applied in most cases. This is because it would

mean that the modeled dependency depicts the domain wrongly, though it has been modeled explicitly.

- Transforming the full-mandatory feature to a selectable feature, e.g. an optional feature. Now the dependency would make sense, though the semantic of the former full-mandatory feature has been changed. This transformation might be applied if this case points to the fact that the modeled full-mandatory feature should not be a full-mandatory feature.
- Changing the feature that is dependent on the full-mandatory feature. Changing means that this feature might change its domain variability or the domain relationships of this feature have to be restructured.

Again it must be mentioned that the anomalies point to a falsely modeled domain. Analysts have to investigate the appearing problems very seriously to capture the domain correctly.

Feature models which contain inconsistencies must be divided in those which contain inconsistencies on the domain level and in those on the product configuration level. To resolve domain inconsistencies as described in section 3.5.1, a thorough analysis is needed, as inconsistencies on this level definitely point to a falsely modeled domain. Inconsistencies on the product configuration level can be easily resolved by obeying the semantics of the used relationships. Therefore, missing features have to be added and additional conflicting features have to be removed.

4. Tool Support

As mentioned in the previous section, adequate tool support is needed to facilitate feature model development and in deriving product configurations from the domain model. Especially a tool should be able to detect inconsistencies in the domain and should indicate deficiencies of the model because of redundancies and anomalies.

At our group we have developed a prototype of a requirements engineering tool for software product lines – named *RequiLine* – that supports feature modeling. This tool allows to create feature models – textual or graphically and to query for specific information, the user is interested in. Furthermore RequiLine provides a consistency checker which is able to detect inconsistencies on the domain and on the product configuration level [13]. RequiLine has been positively evaluated by a small local software company and by a global player of the automotive industry. In both cases, RequiLine helped to detect inconsistencies in the domain model and in product configurations. Until now, the process of product configuration is only weakly supported, as the user has to manually select which features should be part of a product and which not. Furthermore, RequiLine is capable of detecting the inconsistencies mentioned in section 3.5 but a product configuration wizard guiding the user through the steps of selecting features would be desirable. This kind of a wizard would avoid creating inconsistent product configurations and would not just detect inconsistencies afterwards, like the consistency checker currently does.

5. Summary and Future Work

In this paper we have illustrated types of deficiencies a feature model might contain. The deficiencies can be redundancies, anomalies and inconsistencies in the domain and the product configuration model. We have defined what is meant by redundancy, anomalies and inconsistency in feature models and we have shown, which cases of redundancy, anomalies and inconsistency can appear and how these cases might be resolved. In most cases the deficiencies cannot be resolved automatically but a serious investigation and analysis by the domain experts is needed. Most deficiencies result from the overlapping semantic of domain relationships and dependencies. Therefore we will evaluate the measure of adapting the meta-model for features, so that the use of dependencies is constrained in a way that no anomalies and inconsistencies can appear and therefore avoiding these cases. The detection of redundancies and inconsistencies is a very hard task in very large models and should therefore be tool supported. RequiLine is a first prototype that is able to detect such inconsistencies and other model deficiencies like feature property completeness and categorizes them into light and severe deficiencies.

Our future work comprises the analysis of the completeness of the identified cases of redundancies and inconsistencies and to analyze relations to approaches that rely on artificial intelligence techniques for product configurations. Furthermore we want to enhance RequiLine in a way that it will be able to detect redundancies and the mentioned anomalies as well, and provide a product configuration wizard.

References

1. Kyo Kang et al., *Feature-Oriented Domain Analysis (FODA) Feasibility Study*, Technical report CMU/SEI-90-TR-021, Software Engineering Institute, Carnegie Mellon University, 2000.
2. Kyo Kang et al., *Concepts and Guidelines of Feature Modeling for Product Line Software Engineering*, 7th International Conference on Software Reuse (ICSR), Austin, Texas, USA, pp. 62-77, April 15-19, 2002.
3. Ulrich Eisenecker, Krzysztof Czarnecki, *Generative Programming – Methods, Tools, and Applications*, Addison-Wesley 2000.
4. Martin L. Griss, John Favaro, Massimo d'Alessandro, *Integrating Feature Modeling with the RSEB*, Proceedings of the Fifth International Conference on Software Reuse, IEEE Computer Society, Los Alamitos, CA, USA, 1998.
5. D. Fey, R Fajta, A. Boros, *Feature modeling: A meta-model to enhance usability and Usefulness*, in SPLC2, LNCS 2379, pages 198–216. Springer, 2002.
6. J. Savolainen et al., *Feature analysis*, Technical report, ESAPS, June 2001.
7. Len. Bass, Paul. Clements, Patrick. Donohoe, John. McGregor, Linda. Northrop, *Fourth Product Line Practice Workshop Report*, CMU/SEI-2000-TR-002, Software Engineering Institute, Carnegie Mellon University, 2000.
8. Horst Lichter, Alexander Nyßen, Thomas von der Maßen, Thomas Weiler, *Vergleich von Ansätzen zur Feature Modellierung bei der Softwareproduktlinienentwicklung*, Aachener Informatik Berichte, Aachen 2003.
9. Arie van Deursen, Paul Klint, *Domain-Specific Language Design Requires Feature*

Thomas von der Maßen, Horst Lichter

- Descriptions*, Journal of Computing and Information Technology, 2001
10. M. Simos, D. Creps, C. Klinger, L. Levine, D. Allemang, *Organization domain modeling (ODM) guidebook 2.0*, Technical Report STARS-VC-A025/001/00, Synquiry Technologies Inc., 1996.
 11. R. N. Taylor, W. Tracz, L. Coglianese, Software Development using domain-specific software architectures, ACM SIGSOFT Software Engineering Notes, 20(5):27-37, 1995.
 12. Klaus Müller, John MacGregor, Eva Geisberger, Jörg Dörr, Frank Houdek, Harbhajan Singh, Holger Wußmann, Hans-Veit Bacher, Thomas von der Maßen, *Einsatz von Features im Softwareentwicklungsprozess*, Technical Report, Abschlussbericht des Arbeitskreises „Features“ im Rahmen der Fachgruppe Requirements Engineering der Gesellschaft für Informatik, To be published.
 13. T. von der Maßen, H. Lichter, *RequiLine: A requirements engineering tool for software product lines*, Proceedings of International Workshop on Product Family Engineering PFE-5, Springer LNCS 3014, Siena, Italy, November 2003
 14. Jörg Dörr, *Guidelines for Inspecting Domain Model Relationships*, Diploma thesis, University of Kaiserslautern in cooperation with Fraunhofer Institute for Experimental Software Engineering (IESE) and Avaya Inc., Kaiserslautern, July 2002