A Critical Analysis of Using Feature Models for Variability Management

Kathrin Berg[1] and Dirk Muthig[2]

[1] University of Pretoria, Computer Science Department,
0002 Pretoria, South Africa
kberg@cs.up.ac.za
[2] Fraunhofer Institute Experimental Software Engineering (IESE),
Sauerwiesen 6, D-67661 Kaiserslautern, Germany
Dirk.Muthig@iese.fraunhofer.de

Abstract. The managing of variability across products in a software product line is one of the most important tasks for successful product line engineering. Due to a large number of publications, feature modeling seems to be a popular approach used for dealing with variability in product lines. Such popularity may lead to the assumption that feature modeling is a universal approach used for the management of variability. We require a universal approach to be consistent and scalable; it should provide traceability between variations at different levels of abstraction and across various development artifacts; and there should be a means for visualizing variability. When critically analyzing the feature model and some methods using it for variability management, it was realized that many uncertainties and ambiguities exist. In this paper, we identify the need for a variability management approach that provides the underlying support for feature modeling by fulfilling all the necessary requirements of a universal approach.

1 Introduction

Product line engineering is an approach that develops and maintains families of products while taking advantage of their common aspects and predicted variabilities [19]. A product line infrastructure is developed based on the commonality and variability that exists between the products in a family. Commonality defines those characteristics that are the same between products, whereas variability implies the opposite, i.e. the differences between the products.

Managing variations across a number of products and at different levels of abstraction is a daunting task, especially when the systems supporting various products are very large, as is common in an industrial setting. With large systems the necessity to trace variability from the problem space to the solution space is evident. Approaches for dealing with this complexity of variability need to be clearly established.

The feature model is one of the most common product line artifacts in literature [2] used by various methods as a means for variability management [3][4][6][8][9][15]. By going back to its origins, we establish what the initial purpose of the feature

model was. Observing its popular use, it could be concluded that feature modeling is a universal approach used for the management of variability in software product lines.

We, however, require that a universal approach for variability management is to be consistent, scalable, provide traceability between points of variation at different levels of abstraction and throughout all development phases, as well as provide a means for visualizing variability. Being skeptical about whether feature models are suitable for variability management, we critically analyze them with regard to these requirements and why they are so widely used.

Through the analysis it is discovered that the central role of feature models in product line engineering is not always clear. Differing versions of feature modeling methods exist and are used by various practitioners. Even though the feature model provides an excellent means for visualizing variability at a certain level of abstraction, it is recognized that approaches using the feature modeling method do not fulfill all the requirements necessary for managing variability of a large product line. Presenting this as a problem, we look towards establishing a unified approach for variability management that provides consistency, scalability and traceability, as well as support for visualization mechanisms such as the feature model.

The following section introduces some of the fundamental concepts of a feature model. Section 3 describes the essential requirements of a universal approach for managing variability across different levels of abstraction. Section 4 shows the analysis results for feature modeling methods. Section 5 presents our conclusions and the last section, future work, briefly describes that we look towards using decision models to define a variability management approach that satisfies all the necessary requirements. The contribution of this paper is in bringing together all the requirements for successful variability management and in identifying those that are currently not met by feature modeling methods. We set the scene for future research in this important area for the software engineers faced with large, industrial-strength product lines.

2 The Origin of Feature Models

Domain analysis [4][8] is an activity for identifying commonalities and managing variabilities between products defined in the product line scope. During domain analysis, a feature model is constructed to display the common and variable features of the products in the product line infrastructure.

A feature, as originally defined in the Feature-Oriented Domain Analysis (FODA) method [8], is “a prominent and distinctive user-visible characteristic of a system”. In [1], a feature is defined as “a logical unit of behavior that is specified by a set of functional and quality requirements”. Another definition from [12] is, “a feature represents an aspect valuable to the customer”.

Features are organized in hierarchical tree-like structures called feature models. The idea of a feature model is conceived from the need to present the external or user-visible characteristics of a system to the users[1], given that the usually modeled internal system functions are of no interest to them [8]. Thus, a feature model is the

[1] The term ‘users’ implies any interested stakeholder or external system.

description of a software system's key capabilities and their inter-relationships, and is used as a means for communication between the software engineer and the system users.

From the definitions above, it can be assumed that feature modeling is just as suitable for single-system development as for product-line development, since in the definition there is no mention of specific application to system families. However, the pressing need for a feature model in single-system development has never before been recognized.

Only in connection with system families or software product lines, has it found a greater importance. This is due to the fact that the amount of possible system features is considerably more to communicate and manage in product line development than it is in that of single systems. Therefore, it can be deduced that feature models mainly exist to communicate the *variable,* prominent and distinctive user-visible characteristic between product members of a product line to a user(s).

In [4], feature modeling is described as being essential in product line engineering, given that reusable software artifacts contain inherently more variability than those in single system engineering, and that it is the key technique for identifying and capturing variability. [18] defines feature modeling as "a conceptual domain modeling technique in which concepts are expressed by their features taking into account feature interdependencies and variability in order to capture the concept configurability".

Variability that is represented in a feature model is realized in subsequent development artifacts as variation points. According to [7], "a variation point identifies one or more locations at which the variation will occur". Variation points allow us to provide alternative implementations of functional or non-functional features and thus may appear at any level of abstraction in the development process, from the requirements specification to architecture design, source code implementation and testing.

For this reason, i.e. that the feature model represents variability between products of a product line, it also seems appropriate to use it as a means for *managing* the variability during development. Many methods exist that do attempt to use feature models for this purpose. Some of these methods and tools are, to name a few, FODA [8], Generative Programming [4], FORM [9], FeatuRSEB [6], FArM [15], and CONSUL [3].

3 Requirements for a Universal Approach

To harvest the full benefits of software product lines, variability, specifically the variation points and the relationships between them, needs to be managed in an appropriate and consistent way across software development phases. That is, independent of the level of abstraction or the product-specific context, variation should be easily managed in the same way.

To better understand and therefore also better manage variability, we need to be able to describe the many variations between products and their relationships at a higher level of abstraction [16]. Many proposals have been made for using feature models to manage the variability that exists between products in a software product line or family of systems.

According to [3], methods for supporting variability management need to consider the following:

- Models expressing commonality and variability to support variability management need to be simple, yet universal;
- Variability must be manageable at all levels of abstraction;
- The introduction of new variability expression techniques should be easily possible.

A universal approach needs to possess four important characteristics when being considered for managing variability. The approach needs to be consistent, scalable, provide traceability between points of variation at different levels of abstraction and throughout all development phases, and provide a means to visualize the variability.

- *Consistency*: Standardization can prevent confusion and the incorrect usage of an approach. Variability should be handled the same way at different levels of abstractions and across development phases. A consistent approach reduces the possibility of errors that might occur when using different methods for managing variability at different levels of abstraction.
- *Scalability*: Whether dealing with only a single component or a large complex system, variability must be easily manageable. It is not sufficient for a method to be successful in managing variability on a small scale, but it becomes too complex to handle on a larger scale.
- *Traceability*: Variability at different levels of abstraction and development phases are associated with each other and need to be linked to simplify evolution and maintenance of a software product line. These multi-dimensional relationships need to be managed appropriately.
- *Visualization*: The visualization of variability and its dependencies between products in a product line promotes understandability and provides an overview thereof.

4 Analysis of Feature Modeling Methods

In order to verify our skepticism towards using feature modeling as a means for managing variability, we analyze the feature modeling methods with regards to the requirements of a universal approach as identified in the previous section, i.e. consistency, scalability, traceability, and visualization. The results are described below.

4.1 Consistency

Originally, the FODA method was the first, of many methods to follow, to identify features and their dependencies and represent them in a feature model. Since then, alterations, enhancements and different notations have been developed and used to make up for some or other shortcoming in the original model [3][6][15].

Today, there exist various differing approaches to feature modeling which makes it difficult to use the model as a standard method for variability management. Several

methods using feature models for variability management do not address variability management at all levels of abstraction. Some approaches only focus on managing variability in the problem space [15], whereas others consider variability management in the solution space [16]. There are a few methods that do address the management of variability across these development spaces, such as in [17], FArM and CONSUL. However, these do not use a consistent approach.

For example, The FArM method mainly focuses on the mapping of features in the problem space to the architectural design elements in the solution space, and does not explicitly mention the management of variability at other levels of abstraction. In the CONSUL tool, feature models are used to represent the problem space in terms of commonalities and variabilities, and a newly developed model, the component family model, is used to describe the solution space.

It is necessary to use a simple, clear and unambiguous model for variability management. Following, are a few examples of more inconsistencies that exist between approaches using feature models.

Relations between Features within the Feature Model. In feature models, features are arranged and linked in a hierarchical tree-like structure. The hierarchical relation between features has different meanings in different approaches. In [12] it has been identified that the hierarchical relation between features is used for refinement, decomposition and as a "requires" relation. It is proposed that features are firstly arranged hierarchically by the composition rules, i.e. the "requires" relation, and thereafter by other relations.

In FODA, features are linked to each other according to aggregation/decomposition, and generalization/specialization relations. "The structural relationship *consists of*, which represents a logical grouping of features, is of interest" [8]. However it is not clear as to which type of relationship is most important.

Feature Categorization. All existing methods seem to agree upon the notion of a mandatory feature. The problem lies with defining the different selections of a set of variable features.

In FODA the term "alternative features" is used to "indicate that no more than one specialization can be made for a system" [8], i.e. from a set of variable features only one must be chosen. In [6] or- and xor-features have been categorized as "variant features". The former defines a selection of one or more features from a set, and the latter mutually exclusives features in a set. In [12], all features are designated as optional, and sets of variable features are assigned multiplicities to enable the relevant feature selection.

Additionally, some have identified the need for other feature categorizations, such as, external features as in [17]. Even higher-level categorizations have been suggested, such as capability, operating environment, domain-technology, and implementation technique features in FODA, or functional, interface and parameter features as in [12].

Relations between Features and Other Development Artifacts. Features are classified at a higher level of abstraction than that of requirements, and can therefore be used as constructs to group related requirements. In [12] it is stated that there is a 1-to-n relationship between requirements and features, where in [17] they say that there is a n-to-n relationship between them, i.e. a requirement may apply to several related features in a set and a particular feature may meet more than one requirement. This inconsistency is described in more detail in section 4.3

Feature Dependencies. Dependencies that exist between features in a feature model are dealt with differently in various methods. In the FODA approach dependencies between features are described as composition rules accompanying the feature diagram, whereas in [12] the dependencies are modeled by connecting the dependant features with each other on the feature diagram self. Another approach deals with feature dependencies in separate diagrams altogether [5].

Graphical Notation. There exist various different or enhanced graphical notations for modeling features and representing them in a diagram. Even though each one differs from the other, some use the conventional notation with circles and arcs [8][4][12], whereas others use a UML based notation [6][17].

4.2 Scalability

Scalability is defined by [20] as, "how well a solution to some problem will work when the size of the problem increases". Thus far, many example systems, used in literature to illustrate the use of feature models in variability management, have been rather small [10][12][16][17] compared to the enormous systems that do exist in organizations.

When applying feature modeling techniques to illustrate variable features and their inter-dependencies of a number of large product-members, the feature model rapidly increases in size to such an extent that it becomes too complex to manage and impossible to keep an overview. The problem usually lies with the many relationships and dependencies that may exist between variable features. Trying to represent them all together results in a convoluted model that is difficult to read and understand.

Attempts have been made to deal with this problem of scalability by adding new context-specific models or adjustments to complement the feature model as in [5] and [15]. These usually only provide a solution within a given context, and are therefore not appropriate for standardization.

Some model various feature diagrams representing smaller sub-domains at different levels of abstraction as in [9] and [17], therefore only having to have to deal with smaller feature model-"views" and not with one large model. This would resolve the complexity and overview issue. However, it is unclear how the relationships between features across different "views" are managed.

4.3 Traceability

Traditionally a decision as to weather a certain characteristic is included in a product or not was made during requirements specification and the software product was designed accordingly. However, with the development of a product line, these decisions need to be delayed as long as possible in the development process, or at least to the point where it is most beneficial to an organization [17]. This means that variability must be considered and managed at each development phase, from the initial requirements to the final implementation.

Traceability provides a link for dependencies between artifacts, amongst others, requirements specification, architecture design, source code, and test plans, created during different phases of product line development. Well established traceability improves the comprehension of the product line infrastructure and its product-members' development, and provides support for their evolution and maintenance [14].

The issue of tracing variability between models at different levels of abstraction is directly related to the feature interaction problem, described by [6] as follows. "The problem is that individual features do not typically trace directly to an individual component or cluster of components – this means, as a product is defined by selecting a group of features, a carefully coordinated and complicated mixture of parts of different components are involved". This implies an n-to-n relationship between features and components, or design elements.

There have been contradicting reports about the relationship between requirements and features in a feature model. In [17] it is mentioned that there is an n-to-n relationship between features and requirements, meaning "that a particular requirement may apply to several features in the feature set and that a particular feature may meet more than one requirement." Whereas [12] and [13] state that there is an n-to-1 relationship between requirements and features. Since features are an abstraction of requirements, there are at least one or more requirements linked to one feature. See figure 1.

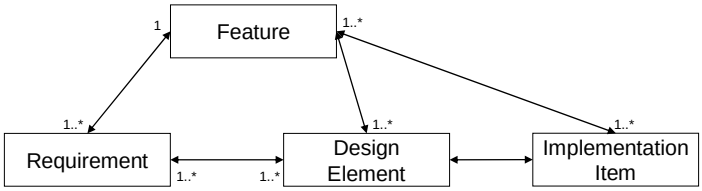

Fig. 1. Relationships between features and other development artifacts (taken from [12])

There are methods that attempt to map features to the architecture or design elements, i.e. FeatuRSEB and FArM. However, the former does not fulfill the requirement of scalability. Both do not address traceability of variability between artifacts other than the feature model and the architecture. In [14], for example, traceability is implemented by using the Javadoc tool to link source code to various other types of documentation. A drawback of this approach is that it is language-dependant, and it

has not yet been established how to link artifacts that are not text-based, such as feature models, to text-based and other artifacts.

In order to manage variability between artifacts at different levels of abstraction, it is necessary for traceability links to represent a 1-to-1 relationship between all artifacts developed in the problem space and the solution space. However, this is hardly possible with the many cross-cutting features, for example security, that exist in and across products. Figure 2 illustrates that in order to achieve this type of relationship between all artifacts, a specific intermediate model is needed that can record the individual relationships and provide the appropriate mappings.

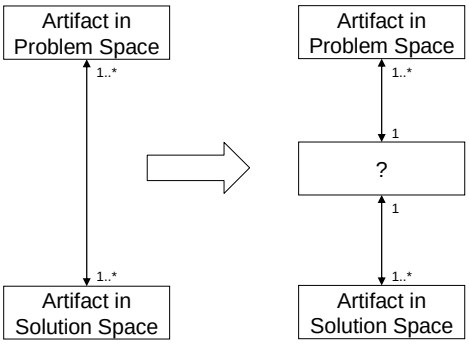

Fig. 2. Relationship between artifacts in problem space and solution space

4.4 Visualization

As already established in section 2, the main role of the feature model was to represent the variability between products in a product line to a user. The graphical representation of hierarchically organizing variable features and their related sub-features, in order to provide an overview and increase understandability of variability, is a good means for visualizing variability.

5 Conclusions

Feature models play a central role in many methods that have been established for variability management. We have identified that the feature modeling approach has a number of weaknesses and inconsistencies, and methods using them for variability management do not satisfy all the needed requirements.

Although the benefit of using the feature model as a means to visualize and communicate software product capabilities to a user is clear, it is not truly suited as a general means for managing variability. Especially where a clear and properly de-

fined approach is necessary, vague definitions such as “a feature is any end-user visible characteristic of a system” or “a feature represents an aspect valuable to the customer“ are not sufficient. Also, as said in [10], “the fuzzy nature of features makes it difficult to formalize its precise semantic, validate results, and provide automated support.”

The fact that there have been many extensions and alterations to the approach shows that it lacks some of that which the user seeks to accomplish when using it. We are not claiming that the feature model has no place in the development of software product lines. However, we do recognize the need for a universal approach that not only provides the means for visualizing variability, but can be consistently used at a generic level, is scalable, and provides traceability between all product line artifacts.

6 Future Work

The conclusions of the previous section have led us to further investigate methods for variability management that fulfill the requirements of a universal approach. Based on the concepts of variability management as described in [1] and [11], we look towards using decision models as a means to capture, describe and manage variability in software product lines across all levels of abstraction and throughout all development phases. The decision model is described as capturing the relationships between variation points. A decision can be described as being a variation point that typically constrains the resolution of other variation points. The role of the decision model is to define how the variant features relate to certain high-level decisions that characterize the distinction between members of the product line. Although not yet formally verified, work is progressing to prove that the decision modeling approach will fulfill the requirements necessary for variability management, and at the same time can be used to visualize variability in similar ways as the popular ones known from feature models.

Acknowledgements

This work was partially funded by the South African Research Foundation Grant No.2196 and the University of Pretoria Postgraduate Study Abroad Bursary Programme. The authors thank Judith Bishop for support of this work and her assistance in improving the paper.

References

1. Atkinson, C., Bayer, J., Bunse, C., Kamsties, E., Laitenberger, O., Laqua, R., Muthig, D., Paech, B., Wüst, J., Zettel, J.: Component-based Product Line Engineering with UML. (Component Software Series) London, Addison-Wesley (2001)

2. Bosch, J. (Editor): Proceedings 2nd Groningen Workshop on Software Variability Management: Software Product Families and Populations (2004)
3. Beuche, D., Papajewski, H., Schröder-Preikschat, W.: Variability management with feature models. Science of Computer Programming. Vol. 53, No. 3 (2004) 333-352
4. Czarnecki, K., Eisenecker, U.W.: Generative Programming. Addison-Wesley (2000)
5. Ferber, S., Haag, J., Savolainen, J.: Feature Interaction and Dependencies: Modeling Features for Reengineering a Legacy Product Line. In: Proceedings of the 2nd Software Product Line Conference. Lecture Notes in Computer Science, Vol. 2379. Springer-Verlag Berlin Heidelberg (2002) 235-256
6. Griss, D., Allen, R., d'Allesandro, M.: Integrating Feature Modelling with the RSEB. In: Proceedings of the 5th International Conference of Software Reuse, ICSR-5 (1998)
7. Jacobson, I., Griss, M., Jonsson, P.: Software Reuse – Architecture, Process, and Organization for Business Success. Addison-Wesley (1997)
8. Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, A.: Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-021, Software Engineering Institute, Carnegie Mellon University, Pittsburgh (1990)
9. Kang, K.C., Kim, S., Lee, J., Kim, K., Shin, E., Huh, M.: FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures. Annals of Software Engineering, 5 (1998) 143-168
10. Kang, K. C., Lee, J., Lee, K.: Concepts and Guidelines of Feature Modeling for Product Line Software Engineering. In: Proceedings of the 7th International Conference on Software Reuse: Methods, Techniques, and Tools. (2002) 62-77
11. Muthig, D.: A Lightweight Approach Facilitating an Evolutionary Transition Towards Software Product Lines. PhD Thesis, Fraunhofer IRB Verlag (2002)
12. Riebisch, M.: Towards a More Precise Definition of Feature Models. In: Workshop at ECOOP. Books On Demand GmbH, Darmstadt, Germany (2003) 64-76
13. Riebisch, M.: Supporting Evolutionary Development by Feature Models and Traceability Links. In: Proceedings 11th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems, ECBS 2004, Brno, Czech Republic (2004) 370-377
14. Sametinger, J., Riebisch, M.: Evolution Support by Homogenously Documenting Patterns, Aspects and Traces. In: 6th European Conference on Software Maintenance and Reengineering, Budapest, Hungary. Computer Society Press (2002) 134-140
15. Sochos, P., Philippow, I., Riebisch, M.: Feature-Oriented Development of Software Product Lines: Mapping Feature Models to the Architecture. In: Proceedings of 5th Annual International Conference on Object-Oriented and Internet-Based Technologies, Concepts, and Applications for a Networked World, NODe 2004. Lecture Notes in Computer Science, Vol. 3263. Springer-Verlag Berlin Heidelberg (2004) 138-152
16. Tekinerdogan, B., Aksit, M.: Managing Variability in Product Line Scoping using Design Space Models. In: Proceedings of Software Variability Management Workshop, Groningen, IWI 2003-7-01, The Netherlands (2003) 5-12
17. van Gurp, J., Bosch, J., Svahnberg, M.: Managing Variability in Software Product Lines. In: Landelijk Architectuur Congres, Amsterdam (2000)
18. Vranić, V.: Reconciling Feature Modeling: A Feature Modeling Metamodel. In: Proceedings of 5th Annual International Conference on Object-Oriented and Internet-Based Technologies, Concepts, and Applications for a Networked World, NODe 2004. Lecture Notes in Computer Science, Vol. 3263. Springer-Verlag Berlin Heidelberg (2004) 122-137
19. Weiss, D. M.; Lai, C. T. R.: Software Product-Line Engineering. A Family-Based Software Development Process. Addison-Wesley (1999)
20. The Free On-line Dictionary of Computing, © 1993-2004 Denis Howe - http://foldoc.doc.ic.ac.uk/foldoc/Dictionary.gz Website accessed: 30 March 2005