Comprehension and Utilization of Core Assets Models in Software Product Line Engineering

Iris Reinhartz-Berger[1], Arnon Sturm[2], and Arava Tsoury[1]

[1] Department of Information Systems, University of Haifa, Haifa 31905, Israel
iris@is.haifa.ac.il, aravabt@gmail.com
[2] Department of Information Systems Engineering,
Ben-Gurion University of the Negev, Beer Sheva 84105, Israel
sturm@bgu.ac.il

Abstract. In software product line engineering, *core assets* are reusable artifacts that are intended to be used by a family of software products in order to improve development productivity and quality of particular software products. In order to support the construction and maintenance of core assets, various modeling methods have been proposed. However, the assessment of these methods is still in an incubation stage. In fact, only several frameworks for comparing and evaluating these methods have been suggested. These mainly refer to lists of criteria whose examination is sometimes subjective and opinion-dependent. In this paper, we call for empirical evaluation of the comprehension and utilization of core assets and report the initial results of a series of studies we performed in this context.

Keywords: variability management, software product line engineering, domain analysis, UML, feature-orientation, evaluation

1 Introduction

In Software Product Line Engineering (SPLE), a *core asset* is "a reusable artifact or resource that is used in the production of more than one product" [5]. The development of core assets intends to improve productivity, increase quality of individual products, decrease development cost, decrease time to market or to launch new products or versions, and enable moving into new markets in shorter times. Core assets have different forms that may be useful in a software production process, one of which is domain models. These models capture both existing commonality and allowed variability of given product lines.

Reviewing 97 papers that describe variability management approaches in SPLE, reported from 1990s to 2007, Chen and Babar [4] conclude that the main corpus of approaches focuses on variability modeling and utilizes feature models (33 works) or UML and its extensions (25 works) for this purpose. Feature-oriented methods, such as [6], [13], [14], and [21], support specifying domain models as sets of characteristics relevant to some stakeholders and the relationships and dependencies among them. Variability is specified in terms of mandatory vs. optional features, alternatives, OR features, 'require' and 'exclude' dependencies among features, feature groups, and composition rules. UML-based SPLE methods (e.g., [10], [18], [20], [25],

and [26]) usually suggest profiles for handling variability-related issues, including specification of mandatory and optional elements, dependencies among elements, variation points, and possible variants. Some UML-based methods suggest extending UML or representing variability aspects orthogonally to "regular" UML models of the product families, e.g., [11].

As the number of suggested modeling methods increases, several evaluation frameworks have been proposed for comparing methods, belonging to different categories, e.g., [9], [12], [15], and [24], or within certain categories, e.g., [8]. Matinlassi [15], for example, refers to four main comparison criteria: context, user, contents, and validation. Haugen et al. [12] suggest examining the ways variability and commonality are modeled, the support for iterative and incremental system family development, and the production of individual systems. Djebbi and Salinesi [8] compare feature-oriented notations in terms of different criteria, including readability, simplicity and expressiveness, adaptability, scalability, and others. Although the various criteria may help understand the benefits and limitations of the different methods, their usage in examining and comparing the methods is limited as they are subjective and usually criticized as opinion-oriented [4].

Despite their amenability to be empirically evaluated, relatively minor attention is allocated for the empirical evaluation of SPLE methods in general and variability management approaches in particular. These studies highlight different aspects in SPLE, including product derivation [22], quality assurance [2, 7], and architecture process activities [1]. In this paper we draw a general evaluation framework for comparing core assets modeling methods. This framework, which refers to both specification and utilization aids, is used for better understanding the sources of difficulties of core assets modeling methods. In a series of three studies, we started examining the specification aids of modeling methods to clearly describe common and variable parts in core assets and the relevant utilization aids, which aim at guiding the developer in generating, deriving, and building valid software products. We report on some sources of difficulties we found in the comprehension and utilization of core assets models using feature-oriented and UML-based methods.

The remainder of this paper is organized as follows. Section 2 reviews the suggested dimensions for evaluation, whereas Section 3 describes two core assets modeling methods on which we conducted the evaluation so far and justifies their selection. Section 4 elaborates on the empirical studies and reports our initial results. Finally, Section 5 concludes and refers to future research directions.

2 Dimensions for Evaluating Core Assets Modeling Methods

When evaluating core assets modeling methods, two important dimensions can be identified: *specification*, which refers to the collection of aids required for specifying both existing commonality and allowed variability in a software product line, and *utilization*, which refers to the different means to use core assets in order to create particular software product in the domain.

The specification aids are further divided into commonality- and variability-related ones. *Commonality-related aids* are used for specifying aspects that all (or most of)

the products in the line exhibit, while *variability-related aids* enable specifying added values that not all the products in the family include in the same way.

The utilization dimension refers to the aids needed in core assets modeling methods in order to improve the effectiveness and efficiency of creating particular product artifacts in certain domains. This includes guidance and validation. *Guidance* refers to the ways in which core assets can be used for specific needs (i.e., in the development or production of particular software products), while *validation* refers to the mechanisms and tools that may be provided by the modeling methods for enabling alignment of specific software products with the domain constraints and rules as specified in core assets.

Table 1 summarizes the main specification and utilization aids of feature-oriented and UML-based methods. In order to define sources of difficulties in specifying and modeling core assets in these categories, we used the suggested evaluation framework and conducted three studies (the focus of each study is depicted in Table 1). Due to the large number of methods in each category, we had to select specific methods for evaluation. Explanations on the selected modeling methods, as well as the reasons for their selection are provided next.

Table 1. Evaluation Framework for Core Assets Modeling Methods

Category	Specification Aids		Utilization Aids	
	Commonality	**Variability**	**Guidance**	**Validation**
Feature-oriented	Mandatory and optional elements, dependencies	Feature groups, alternatives, and OR-related features	Cardinality, rationale, constraints	Instantiation and configuration conformance
UML-based	Mandatory and optional elements, dependencies	Variation points, variants	Cardinality, openness, reuse mechanisms, binding time	Specialization and configuration conformance

Legend: Eval 1, Eval2, Eval 3

3 The Selected Modeling Methods: CBFM and ADOM

3.1 Feature-Oriented Methods and CBFM

In feature-orientation, ***features*** are defined as end-user characteristics of systems, or distinguishable characteristics of concepts that are relevant to some stakeholders of the concepts [13]. Features can be composed and decomposed into trees, where the edges represent dependencies between features. Some of the feature-oriented methods, such as [14], concentrate on commonality specification and do not explicitly specify variability. However, guidance is partially supported in these methods, mainly

via XOR and OR constructs or via explicit textual constraints and guidelines. Some methods, e.g., [6] and [23], support representing variation points and variants via feature groups and refer to guidance via OR and XOR constructs.

Cardinality-Based Feature Modeling (CBFM) [6] exceeds the expressiveness of other feature-oriented methods by enabling usage of OCL for specifying different dependencies and allowing definition of various cardinalities for better guiding the development of particular software products. In particular, CBFM extends the expressiveness of feature diagrams in FODA [13], the ancestor of most feature-oriented methods, with five main aspects: (1) *cardinality*, which denotes how many clones of a feature can be included in a concrete product, (2) *feature groups*, which enable organizing features and defining how many group members can be selected at certain points, (3) *attribute types*, indicating that attribute values can be specified during configuration, (4) *feature model references*, which enable splitting a feature diagram into different diagrams, and (5) *OCL constraints*.

3.2 UML-based Modeling Methods and ADOM

Most UML-based methods model commonality-related aspects via dedicated stereotypes for differentiating mandatory (sometimes called kernel) and optional elements. Some works explicitly specify variability using both «variation point» and «variant» stereotypes, while others specify only one of these concepts and the other is implicitly specified from its relationships with the other concept.

We selected the Application-based DOmain Modeling (ADOM) method [18] for our evaluation, since it consistently integrate all the main stereotypes from other methods in the UML-based SPLE category [19] and it explicitly refers to guidance and validation of software products with respect to core assets, aspects which other methods in this category tend to neglect. Furthermore, it enables explicit specification of both variation points and variants and it allows specifying ranges of multiplicity.

At the basis of ADOM there is a profile that includes the following six stereotypes: (1) «multiplicity», specifying the range of product elements that can be classified as the same core element, (2) «variation point», indicating locations where variability may occur, including rules to realize the variability in these locations, (3) «variant», which refers to possible realizations of variability and is associated to the corresponding variation points, (4) «requires» and (5) «excludes», which determine dependencies between elements (and possibly between variation points and variants), and (6) «reuse», which is used for guiding the developer about the possible usages of the core asset element in specific products.

4 Empirical Evaluation of Core Assets Modeling Methods

4.1 Eval1: Comprehension and Utilization of ADOM's Models

The first study was associated with two research questions: (1) Are the specification aids of ADOM well understood and to what extent? (2) Are the specification aids of

ADOM well utilized and to what extent? The subjects of this study were 15 advanced undergraduate and graduate students in an Information Systems program at the University of Haifa, Israel, who took a seminar course named "Advanced Topics in Software Engineering" in 2009. During the course, the students studied domain engineering techniques, focusing on ADOM and its capabilities. The study took place towards the end of the course as a class assignment. The students got a domain model (in ADOM) and had to answer questions in three categories. In the first category of questions, which referred to *comprehension*, the subjects had to answer 14 true/false questions regarding the domain and explain their answers based on the given model. The questions referred to both commonality and variability aspects. The second group of questions, *validation*, required finding violations in a particular application, with respect to the domain constraints as specified in the given model. For checking this task, we prepared a list of 9 mistakes (or inaccuracies) in the application model and measured the performance of the subjects in terms of precision, recall, and F-measure. Finally, in the third part, *guidance*, the subjects were asked to model another application in the domain based on a list of requirements and the given domain model (in ADOM). In this part, we examined how the specification aids were utilized for guiding the creation of particular models.

The results of this study brought up the following main points. First, variant-related aspects are better comprehended than variation point-related aspects. Our conjecture regarding this observation is that variation points are more abstract, usually refer to several elements (variants) and include information regarding the way to realize the variability. Thus, their specification is more difficult to understand than that of variants, which are more concrete and focus on particular elements. Second, errors that referred to commonality-related aspects, including such that refer to optional elements and not just to mandatory ones, are easier to find than errors that referred to variability. Furthermore, variability-related errors that involved several different model elements were the most difficult to detect (only two students found one such error each). Third, the subjects had difficulties in mapping the particular application elements to the domain elements as specified in the domain model. As this mapping may reveal anchors for validation, these difficulties also prevent the subjects from correctly identifying problems that are related to both commonality and variability issues. Finally, we found a correlation between the success in applying a variation point and the success to utilize its variants. However, it seems that the guidance provided by variation points is less considered than the guidance provided by the variants. A possible reason for this may be again the different abstraction levels of variation points and variants.

4.2 Eval2: Specification and Guidance Aids in ADOM

The second study addressed two research questions: (1) Do variability specification and guidance aids help comprehend core assets and to what extent? (2) Do variability specification and guidance aids help create or model correct products and to what extent? The subjects of this study were 116 advanced undergraduate students in an Information Systems Engineering program at Ben-Gurion University of the Negev, Israel, who took a mandatory course named "Object-Oriented Analysis and Design"

in 2009. During the course, the students studied the ADOM method and the study took place as part of the final exam in the course. The students were randomly divided into four groups, each of which got a core asset model (in ADOM) and had to answer 15 comprehension questions regarding the given model and to model a particular application in the domain. The model given to the first group included only commonality-related stereotypes, namely «multiplicity», «requires», and «excludes». The model given to the second group included guidance-related stereotypes (i.e., «reuse») besides the commonality-related stereotypes, while the model given to the third group included, besides the commonality-related stereotypes, variability-related stereotypes, namely «variation point» and «variant». The model given to the fourth group included all six stereotypes. Despite the difference in the sets of stereotypes provided to the four groups, the models included similar (equivalent) information using UML expressiveness and associating textual notes when required.

The following interesting points have risen from this study. First, the guidance aids, which explicitly explain how to reuse a core asset element in a particular software product, help comprehend aspects that refer to commonality and variability issues, and not just to reusability. This was especially remarkable when referring to variation points and their rules to select variants. Our conjecture is that explicit specification of guidance required additional attention from the students, thus resulting in better outcomes. Second, the existence of all stereotypes seemed to complicate the core asset models and negatively affect comprehension. Finally, no statistically significant differences were found among the particular models produced by the various groups from the core asset. We believe that this is due to the clear and unambiguous requirements of the requested system, a situation which is less realistic in "real life", but required for a controlled experiment.

4.3 Eval3: Comprehension of CBFM and ADOM Specification Aids

The research question in the third study was: The specifications of which method, out of CBFM and ADOM, are more comprehensible and to what extent? The subjects in this study were 18 advanced graduate and undergraduate information systems students at the University of Haifa, Israel who took the seminar course “Advanced Topics in Software Engineering” in 2010. During the course, the students studied various domain engineering techniques, focusing on CBFM and ADOM and their ability to specify core assets. The study took place towards the end of the course as a class assignment. The students were equally divided into two groups of 9 students each according to their grades in relevant modeling courses, their academic and industrial background, and their familiarity with the examined methods. The students in the first group got a CBFM model of a domain and a dictionary of terms in that domain, while the students in the second group got an ADOM model of the same domain and the same dictionary of domain terms. The students in the two groups were asked to answer 15 true/false comprehension questions and to provide full explanations to their answers.

The results showed that CBFM outperformed ADOM in commonality-related questions, while ADOM outperformed CBFM in variability-related questions. This outcome is reasonable, as feature-orientation concentrates on a common kernel and its

possible configurations, while ADOM treats software products and product lines as belonging to two different abstraction levels and allows more variability among products that belong to the same product line (e.g., via specialization and extension). Nevertheless, a statistical analysis showed that there is significant difference only in the variability specification and this is in favor of ADOM [19]. In all other categories no statistical significance was found. Still, according to the achieved averages, the overall comprehensibility in ADOM was better than that in CBFM. This outcome somehow questions the widespread opinion [20] that feature-orientation is simpler and, thus, more comprehensible to different stakeholders involved in SPLE and worth further investigation in the future.

5 Summary and Future Work

Different core assets modeling methods have been suggested. These methods are usually evaluated and compared subjectively, using different lists of criteria that highlight various aspects of core assets specification and utilization. We used a different approach for comparing these methods: empirical evaluation of comprehending and utilizing their resultant models. Based on a series of three studies, we noticed that variability is comprehensible and utilizable to a limited extent and that the main source of problem is in comprehending variation points. Yet, when providing explicit guidelines, the comprehension of the domain model increases.

The three conducted studies were relatively limited in their subjects' qualifications, the numbers of participating subjects, required tasks, examined methods, and provided models. Thus, in the future, we plan to replicate the empirical study on larger classes of trained domain engineering students and software developers and to use the suggested framework for comparing and evaluating core assets modeling methods in other categories, such as in Domain-Specific Languages [16].

References

1. Ahmed, F. and Capretz, L. F. The software product line architecture: An empirical investigation of key process activities. Information and Software Technology 50, pp. 1098–1113, 2008.
2. Bagheri. E. and Dasevic, G. Assessing the Maintainability of Software Product Line Feature Models using Structural Metrics. Software Quality Journal, Springer, DOI: 10.1007/s11219-010-9127-2, 2011.
3. Borba, C. and Silva, C. A Comparison of Goal-Oriented Approaches to Model Software Product Line Variability. ER'2009 workshops. LNCS 5833, pp. 244-253, 2009.
4. Chen, L. and Babar, M. A. A systematic review of evaluation of variability management approaches in software product lines. Information and Software Technology 53, pp. 344-362, 2011.
5. Clements, P. and Northrop, L. Software Product Lines: Practices and Patterns. Addison-Wesley, 2002.
6. Czarnecki, K. and Kim, C.H.P. Cardinality-Based Feature Modeling and Constraints: A Progress Report. In Proceedings of the OOPSLA Workshop on Software Factories, 2005.

7. Denger, C. and Kolb, R. Testing and Inspecting Reusable Product Line Components: First Empirical Results. Proceedings of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering, ACM, pp. 184–193, 2006.
8. Djebbi, O. and Salinesi, C. Criteria for Comparing Requirements Variability Modeling Notations for Product Lines. The Fourth International Workshop on Comparative Evaluation in Requirements Engineering (CERE'06), in conjunction with RE'06, 2006.
9. Frakes, W.B. and Kyo, K. Software Reuse Research: Status and Future. IEEE Transactions on Software Engineering, 31 (7), pp. 529-536, 2005.
10. Gomaa, H. Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures, Addison-Wesley Professional, 2004.
11. Halmans, G. and Pohl, K. Communicating the Variability of a Software-Product Family to Customers. Software and Systems Modeling 2 (1), pp. 15-36, 2003.
12. Haugen, Ø. Møller-Pedersen, B., and Oldevik, J. Comparison of System Family Modeling Approaches. Software Product Lines Conference. LNCS 3714, pp. 102-112, 2005.
13. Kang, K., Cohen, S., Hess, J., Novak, W., and Peterson, A. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, 1990.
14. Kyo, C. K. , Sajoong, K., Jaejoon, L., Kijoo, K., Euiseob, S. and Moonhang, H. FORM: A feature oriented reuse method with domain-specific reference architectures. Annals of Software Engineering 5 (1), pp. 143-168, 1998.
15. Matinlassi, M. Comparison of Software Product Line Architecture Design Methods: Comparison of Software Product Line Architecture Design Methods: COPA, FAST, FORM, KobrA and QADA. Proceedings of the 26th International Conference on Software Engineering (ICSE'04), 2004.
16. Mernik, M., Heering, J., and Sloane, A. M. When and How to Develop Domain-Specific Languages. ACM Computing Surveys (CSUR) 37 (4), pp. 316-344, 2005.
17. Ramesh, V. and Topi, H. Human Factors Research on Data Modeling: A Review of Prior Research, an Extended Framework and Future Research Directions, Journal of Database Management, Vol. 13 (2), pp. 3-19, 2002.
18. Reinhartz-Berger, I. and Sturm, A. Utilizing Domain Models for Application Design and Validation. Information and Software Technology, 51(8), pp. 1275-1289, 2009.
19. Reinhartz-Berger, I. and Tsoury, A. Experimenting with the Comprehension of Feature-Oriented and UML-Based Core Assets. Lecture Notes in Business Information Processing (LNBIP) 81, pp. 468–482, 2011.
20. Robak, S., Franczyk, B., Politowicz, K., Extending the UML for modeling variability for system families. International Journal of Applied Mathematics and Computer Science 12 (2), pp. 285-298, 2002.
21. Silva, C., Alencar, F., Araújo, J., Moreira, A., Castro, J. Tailoring an Aspectual Goal-Oriented Approach to Model Features. 20th International Conference on Software Engineering and Knowledge Engineering (SEKE'2008), pp. 472-477, 2008.
22. Sinnema, M. and Deelstra, S. Industrial validation of COVAMOF. Journal of Systems and Software 81 (4), pp. 584-600, 2008.
23. Svahnberg, M., Van Gurp, J., and Bosch, J. A Taxonomy of Variability Realization Techniques. Software – Practice & Experience 35 (8), pp. 705-754, 2005.
24. Trigaux, J.C. and Heymans, P. Modeling variability requirements in Software Product Lines: A comparative survey. Technical report PLENTY project, Institut d'Informatique FUNDP, Namur, Belgium, November 2003.
25. Webber, D. and Gomaa, H. Modeling variability in software product lines with variation point model. Science of Computer Programming, Vol. 53, pp. 305-331, 2004.
26. Ziadi, T., Hélouët, L., and Jézéquel, J.M. Towards a UML Profile for Software Product Lines. Software Product-Family Engineering (PFE'2004), LNCS 3014, pp. 129-139, 2004.