

COSVAM: A Technique for Assessing Software Variability in Software Product Families

Sybre Deelstra, Marco Sinnema, Jos Nijhuis, Jan Bosch

Department of Computing Science and Mathematics, Rijksuniversiteit Groningen,

P.O. Box 800, 9700 AV Groningen, The Netherlands

{s.deelstra|m.sinnema|j.a.g.nijhuis|j.bosch}@cs.rug.nl

Abstract

Evolution of variability is a key factor in the successful exploitation of commonalities in software product families. Assessment of variability can be used to determine how the variability provided by a product family should evolve. In this paper, we present COSVAM (COVAMOF Software Variability Assessment Method), a variability assessment technique that specifically addresses evolution of variability. We exemplify our approach with the Daolian case study.

1. Introduction

Software product families [2][6] are concerned with exploiting commonalities among a set of products, while managing the differences between them. A product family typically consists of a shared architecture, a set of components and a set of products. During product derivation, each product family member derives its architecture from the product family architecture, and selects and configures a subset of the product family components. In addition, each product usually contains some product specific code.

The predominant challenge, in most software product families, is the management of the variability required to facilitate the product differences. As functionality and quality required for products in the family continuously changes due to, for example, changing markets, and advances in technology, the variability in the product family architecture and components anticipates differences in *space* (different products) and *time* (different versions of products). The provided variability, however, is influenced by the accuracy of predictions, as well as the feasibility of incorporating all predicted requirements. Consequently, once the product family is in place, at some point in the lifecycle, evolution will force the product family to handle new functionality and thus previously discarded or unforeseen differences.

Due to the change of differences and the dependence of reuse on variability, we formulate a variability law using the words of Lehman's law on software evolution [11]: "*Variability has to undergo continual and timely change, or a product family will risk losing the ability to effectively exploit the similarities of its members*". A technical problem is how to enact this variability law. The key question in this context is: "how can we determine whether, when, and how variability should evolve?" We refer to activities that deal with answering this question as *assessing software variability during product family evolution*.

As product families in industry have been widely adopted and evolve constantly, most of those organizations perform some form of variability assessment. However, software variability assessment is often done in an ad-hoc and unstructured fashion in the heads of, typically, software architects. The main problem resulting from the ad-hoc assessment is that it mostly leads to product specific adaptations that neglect the benefits of a software product family. In addition, changes may lead to incompatibility with the asset base of the software product family.

Existing literature does recognize the importance of assessing variability [5][6]. Currently, however, no technique for software variability assessment exists and existing assessment techniques typically address only one layer of abstraction, e.g. the software architecture.

The main contribution of this paper is that we present a technique, COSVAM (COVAMOF Software Variability Assessment Method), which resolves the aforementioned concerns. The COSVAM technique consists of five main steps, i.e. set assessment goal, specify provided variability, specify required variability, evaluation of mismatches between provided and required variability and, finally, interpretation of assessment results. It exploits the COVAMOF (ConIPF Variability Modeling Framework) [13] to represent both required and provided variability, and uses the representations to evaluate the mismatches between these.

The remainder of this paper is organized as follows. In the next section, we set the context for our variability assessment technique. In section 3, we discuss related work, and in section 4 we provide a description of the case study at Dacolian BV. Section 5 presents COSVAM and illustrates the technique using the Dacolian case. Finally, the paper is concluded in section 6.

2. Context

In order to be able to present the concepts of variability assessment in a readable and understandable manner, we first need to ‘set the scene’. In this section, we introduce the notion of variation points, variability dependencies, and variability assessment.

Variation points. Variation points identify locations at which variation occurs [10]. These identifiers have been recognized as central elements in managing variability, as they facilitate systematic documentation and traceability of variability, development for and with reuse [4], assessment and evolution.

Variation points can exist at all abstraction layers of a product family, such as features, architecture and component implementations. Variation points in one abstraction layer can *realize* the variability in a higher abstraction level, e.g. an optional architectural component that realizes the choice between two features in the feature graph. Each variation point is associated with zero or more variants and can be categorized to five basic types, i.e. *optional* (zero or one variant out of 1..m associated variants), *alternative* (1 out of 1..m), *optional variant* (0..n out of 1..m), *variant* (1..n out of 1..m), and *value* (a value that can be chosen within a predefined range).

Different realization techniques for implementing variability impose a *binding time* on the associated variation point, i.e. the phase in the lifecycle at which the variation point is bound. Examples of such realization techniques include macros, parameterization, conditional compilation, and dynamically linked libraries (see also, e.g. [10][14]).

Dependencies. Dependencies in the context of variability are restrictions on the variant selection of one or more variation points and originate, amongst others, from the application domain (e.g. customer requirements), implementation details, or restrictions on quality attributes. Although dependencies are often expressed in terms of simple dependencies, in many cases dependencies affect a large number of variation points. In addition, dependencies often can not be stated formally or exact, but have a more informal character, such as “these combinations of parameters will have a negative effect on performance”.

Variability assessment. The situation we address in this paper involves assessment in the context of an evolving product family, viz. a product family with an existing reuse infrastructure and a number of products in the field. Variability in a such a product family is *provided* in terms of variation points with associated mechanisms, dependencies, constraints, binding times and variants. Within a certain timeframe, new product family members pose requirements on the variability of a product family. Examples of such requirements are when a new product family member requires a variation point to enable optional behavior, or when different members require different feature alternatives. This is what we refer to as *required variability*: the variability that is demanded as necessary by new product family members within a certain timeframe.

In [3], it was identified that variability in software evolves according to a number of patterns, such as adding a new variant or variation point, and changing the binding time. These patterns are applied when a *variability mismatch* occurs, in other words, when the required variability does not match the provided variability. These mismatches are expressed in terms of those patterns, for example, ‘change of binding time required’. Variability assessment involves identifying these mismatches, as well as determining the impact of necessary changes.

3. Related Work

Variability assessment is related to a number of existing approaches in the product family domain:

FAST. In their book on a generative approach to product family engineering [15], Weiss and Lai recognize the importance of analyzing variability, as well as predicting changes in the context of a product family. When it comes to actually determining changes to variability, however, the book lacks precision. In the authors’ own words (p. 198): “... You can adapt standard change management techniques to FAST projects, so the FAST PASTA model does not elaborate on those aspects in any great detail”.

Investment analysis. In the PulSE-ECO approach for determining the product family scope [8], DeBaud and Schmid identify that [12] and [16] center on a predefined and limited set of criteria for planning and scoping a product family, and provide a more general approach. Each of the aforementioned approaches, however, is based on rough estimates for implementing characteristics, which assume that the effort for implementing a characteristic is independent from implementing other characteristics.

SEI PLP. In the SEI’s Product Line Practices and Patterns book [6], Clements et al. quote evaluation of variation points with respect to appropriateness, flexibility, and performance as an example of an important

evaluation. The book, however, does not present a detailed technique to perform these evaluations, but suggests modifying existing architecture assessment methods to accomplish this goal (pp. 77-83).

Assessment. Assessments typically follow a standard process consisting of five steps, i.e. goal specification, specification of the provided aspect, specification of the required aspect, analyzing the difference between the provided and required aspect, and interpreting the results. A number of assessment techniques use scenarios to capture the required aspects. Examples of these approaches are ATAM [5], ALMA [1], and SALUTA [9]. These three approaches respectively assess trade-offs between quality attributes, maintainability, and usability in software architectures. The approaches focus on analyzing the architecture of single systems, rather than addressing a software product family for all its abstraction layers.

4. Case Study: Dacolian B.V.

Dacolian B.V. is an SME in the Netherlands that mainly develops intellectual property (IP)-software modules for intelligent traffic systems (ITS). In our case study we focused on the Intrada® product family whose members use outdoor video or still images as most important sensor input (see <http://www.dacolian.nl>).

Dacolian maintains a product family architecture and in-house developed reusable components that are used for product construction. The infrastructure contains special designed tooling for Model Driven Architecture (MDA) based code generation, software for module calibration, dedicated scripts and tooling for product, feature and component testing, and large data repositories. These assets capture functionality common to either all Intrada products or a subset of Intrada products.

Binding is typically done by configuration and license files at start-up time, as well as MDA code generation, pre-compiler directives and Make dependencies for pre-deployment phases.

5. COSVAM

To address the issues discussed in section 2 and 3, we developed a technique called COSVAM (COVAMOF Software Variability Assessment Method). Adopting the standard steps of scenario-based assessment approaches, the COSVAM technique is structured as follows:

1. Set assessment goal: Software variability assessment can be used for a number of goals. Each of these assessment goals influences other steps in the assessment. Before initiating the assessment, the goal therefore has to be clearly demarcated. For our variability assessment

technique, COSVAM, we identify the following five goals:

Goal 1. Determine the ability of the product family to support a new product.

Goal 2. During product derivation, determine whether missing features should be implemented in product specific artifacts or integrated in the product family.

Goal 3. Collecting input data for release planning.

Goal 4. Assess the impact of new features that cross-cut the existing product portfolio.

Goal 5. Determine whether all provided variability is still necessary.

2. Specify provided variability: The second step in COSVAM is to specify the provided variability of the product family on all layers of abstraction. To this purpose, we developed, as part of the ConIPF project [7], the COVAMOF (ConIPF Variability Modeling Framework). The COVAMOF consists, among others, of the COVAMOF Variability View (CVV). The CVV provides a view on the variability provided by the product family artifacts in terms of first-class variation points and first-class dependencies (see Fig. 1). Due to limited space, we only present a brief description of first-class variation points in this framework. More details on, for example, first-class dependencies are found in [13].

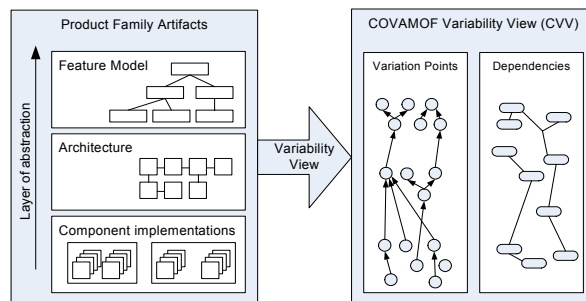


Figure 1. The COVAMOF Variability View (CVV) on the provided variability of several layers of abstraction.

Variation points in the CVV are a view on the variation points in the product family. Each variation point in the CVV is associated to an artifact in the product family, e.g. a feature tree, a feature, the architecture, or a C header file. The CVV distinguishes between five types of variation points, i.e. optional, alternative, optional variant, variant, and value (see also section 2). Fig. 2 presents the graphical representation of these types. Variation points in the CVV contain, amongst others, a description of the *rationale* behind the binding of variants, as well as the associated *binding time* (where applicable). *Realization relations* between variation points in the CVV define how a selection of one or more variation points directly depends on the selection of one or more variation points in a higher level of abstraction in the product family.

Example. To illustrate the provided variability in terms of variation points and realization in the CVV, we present an example from the case study. The Intrada products of version 4.0.0 can be configured to recognize license plates of the Netherlands (NL), Belgium (B) or Germany (D). This feature is expressed by the alternative variation point VP1, where each variant represents one country. At a lower level, VP1 is realized by VP2, VP3, and VP4. The variant variation point VP2 is the selection of different neural network component implementations, the optional variant variation point VP3 is the selection of template matching-based character recognition variants (matchers) and the variant variation point VP4 is the choice between three syntax modules. This example is visualized in Fig. 2.

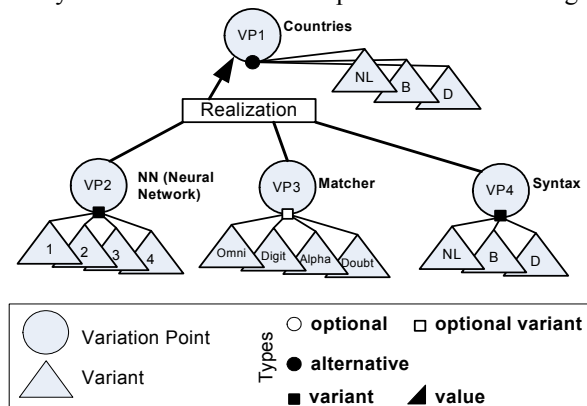


Figure 2. Example of provided variability in the CVV from the case study. The realization relation depicts how the variation points at lower levels realize variation points at a higher level.

3. Specify required variability: The third step in COSVAM is the specification of required variability. This specification captures the variability that is required to accommodate the combinations of functionality and quality in the set of (new) product family members that is used for the assessment. The precise nature of the set depends on the goal. For the first goal, for example, the set of family members consists of one, while for the third goal (release planning), the set consists of both existing and future product family members.

As considerable parts of the required variability will match the provided variability of a product family, it can be specified, amongst others, in terms of provided variants that need to be bound in the individual product family members, plus a ‘delta’ in terms of new variation points and changes to attributes of existing variation points. We therefore also use COVAMOF Variability Model to specify required variability. To this purpose we extended COVAMOF to allow for specification of, for example, new required variation points and variants, as well as earliest and latest allowed binding time. The extended view is referred to as the COVAMOF Required Variability View (CRVV).

As the required variability can be specified in terms of the provided variability, the starting point of the CRVV is a copy of the provided variability view of the product family (CVV). Subsequently, the required functionality and quality of each individual product family member is incrementally integrated into this model, by specifying the delta (e.g. new required variation points and variants), and marking the required variants. We illustrate the resulting specification with an example from the case study.

Example. An example of specifying the required variability as a result of the characteristics of four product instances in the Intrada product family is illustrated in Fig. 3. These products require a new variation point for selecting the recognized country with NL, B and D as new variants.

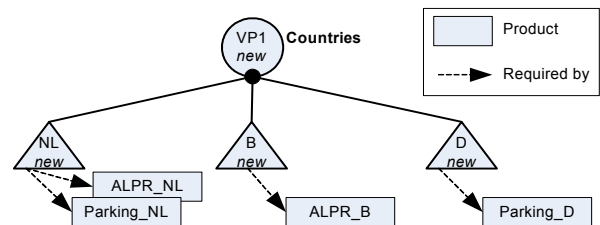


Figure 3. The COSVAM specification of required variability in an earlier release of the Intrada product family.

4. Evaluation of mismatches between provided and required variability: Once the required variability is specified, the next step is to compare the required variability to the provided variability. This step breaks down into two steps, i.e. *identification of variability mismatches*, and *impact analysis*. The identification of variability mismatches involves identifying where conflicts occur between the variability that is required by the set of product family members, and the variability provided by the product family.

Impact analysis is associated with determining what changes are required to resolve the variability mismatches, and what the effects of these changes are on the product family. As mismatches can often be solved in different ways, impact analysis furthermore determines the impact of various realization scenarios, as well as conflicts between mismatches. Impact analysis in part relies on expert judgment, and uses the realization relations of the provided variability to identify how variability at higher layers of abstraction is related to lower layers of abstraction. Evaluation and specifying required variability have to be iterated, as impact analysis may result in new required variability.

5. Interpreting the assessment results: The evaluation step provides information to a designer to decide to which extent variability of the product family supports the required variability. It also describes the impact of various

realization scenarios. During the interpretation stage one of these scenarios is selected.

Example. Early releases of the Intrada product family offered no variability with respect to the useable countries, and all country specific processing was handled product specifically. As Dacolian's products hit the international market, the number of requested countries increased dramatically over a short period of time. Soon, it became unworkable to realize each new country for each separate product. It was decided to move the product specific country variations into the Intrada product family (see Fig. 4).

At this stage the Intrada product family provided variability that allowed the selection of one specific country. The actual binding of the country variation point was at compile time. The binding time lead to a vast number of *different* product instances that became impractical to handle. Next to that, handling multiple countries at the same time became a desired product feature. Assessment of the required variability revealed that there was a mismatch in binding time. To prevent an explosion of product instances it was clear that the binding time should be at run-time.

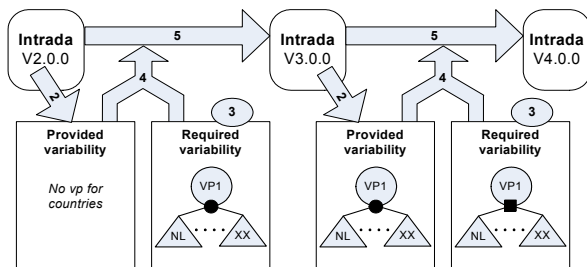


Figure 4. A graphical representation of the evolution process of the country variation point. The numbers in the arrows refer to the steps of the variability assessment. The transition to the new release (5) is based on the evaluation (4) of the mismatch between provided (2) and required (3) variability.

6. Conclusion

In this paper we presented the COVAMOF Software Variability Assessment Method (COSVAM). The contribution of the COSVAM is that it provides a technique for variability assessment for evolution of software product families, where no techniques were available up to date. In this paper, the focus is on the specification of provided and required variability. Currently, we are extending the evaluation step in the method as well as the interpretation of the evaluation results. Finally, although the technique has been applied on the case provided by Dacolian B.V., we intend to apply it on several additional cases in the future.

Acknowledgements. This research has been sponsored by ConIPF [7] (Configuration in Industrial Product Families), under contract no. IST-2001-34438.

7. References

- [1] Bengtsson, P.O., Lassing, N., Bosch, J., van Vliet, H., "Architecture-level Modifiability Analysis (ALMA)", *Journal of Systems and Software*, Vol. 69(1-2), January 2004, pp. 129-147.
- [2] Bosch, J., *Design and Use of Software Architectures: Adopting and Evolving a Product Line Approach*, Pearson Education (Addison-Wesley & ACM Press), ISBN 0-201-67494-7, 2000.
- [3] Bosch, J., Florijn, G., Greefhorst, D., Kuusela, J., Obbink, H., Pohl, K., "Variability Issues in Software Product Lines", *Proceedings of the Fourth Int. Workshop on Product Family Engineering (PFE-4)*, Spain, 2001, pp. 11-19.
- [4] Clauss, M., "Modeling variability with UML", GCSE 2001 - Young Researchers Workshop, Germany, September 2001.
- [5] Clements, P., Kazman, R., Klein, M., *Evaluating Software Architectures*, Methods and Case Studies, Addison-Wesley, ISBN 0-201-70482-X, 2001.
- [6] Clements, P., Northrop, L., *Software Product Lines: Practices and Patterns*, SEI Series in Software Engineering, Addison-Wesley, ISBN: 0-201-70332-7, 2001.
- [7] ConIPF, The ConIPF project (Configuration of Industrial Product Families), <http://segroup.cs.rug.nl/conipf>.
- [8] DeBaud, J.M., Schmid, K., "A systematic approach to derive the scope of software product lines", *Proceedings of the 21st Int. Conf. on Software Engineering*, California, USA, May 1999, pp. 34-43.
- [9] Folmer, E., Gulp, J., Bosch, J., "Architecture-Level Usability Assessment", accepted for EHCI, Hamburg, July 2004.
- [10] Jacobson, I., Griss, M., Jonsson, P., *Software Reuse. Architecture, Process and Organization for Business Success*, Addison-Wesley, ISBN: 0-201-92476-5, 1997.
- [11] Lehman, M.M., Ramil, J.F., Wernick, P.D., Perry, D.E., Turski, W.M., "Metrics and Laws of Software Evolution - The Nineties View", *Proceedings of the Fourth International Software Metrics Symposium*, Albuquerque NM, USA, November 1997.
- [12] Robertson, D. Ulrich, K., "Planning for product platforms", *Sloan Management Review*, Vol. 39 (4), 1998, pp. 19-31.
- [13] Sinnema, M., Deelstra, S., Nijhuis, J.A.G., Bosch, J., "COVAMOF: A Framework for Modeling Variability in Software Product Families", accepted for the 3rd Software Product Line Conference, Boston, USA, 2004.
- [14] Svahnberg, M., Gulp, J. van, Bosch, J., "A Taxonomy of Variability Realization Techniques", ISSN: 1103-1581, Blekinge Institute of Technology, Sweden, 2002.
- [15] Weiss, D.M., Lai, C.T.R., *Software Product-Line Engineering: A Family Based Software Development Process*, Addison-Wesley, ISBN 0-201-694387, 1999.
- [16] Whitey, J., "Investment analysis of software assets for product lines", Technical report CMU/SEI-96-TR-010, Software Engineering Institute, 1996.