Managing the complexity of variability in software product families

Deelstra, Keimpe Sybren; Sinnema, Marco

PART IV. Evolution

The last part of our variability management framework is directed towards evolution of product families. As we noted in the Introduction to this thesis, an alternative approach to decreasing application engineering cost is to make sure there are less mismatches between the variability provided by a product family and the variability required by the products. This Part presents the background, contents, and experiences of applying the COVAMOF Variability Assessment Method (COSVAM). COSVAM is the first technique for assessing variability with respect to the needs of a set of product scenarios. The five steps of COSVAM (identify assessment goal, specify provided variability, specify required variability, evaluate variability, interpret assessment results) form a structured technique that can be tuned to address a variety of situations where the question of whether, how and when to evolve variability is applicable.

Chapter 11 Variability Assessment

An important aspect of software variability management is the evolution of variability in response to changing markets, business needs, and advances in technology. In Chapter 5, we discussed that the evolution of variability should make sure it prevents mismatches between variability provided by the product family, and the variability required by the products. Variability assessment is a technique that addresses this aspect. This chapter explains what variability assessment is, and what the issues are in the current practice.

Based on	Section numbers
S. Deelstra, M. Sinnema, J. Nijhuis, J. Bosch, COSVAM: A Technique for Assessing Software Variability in Software Product Families, Proceedings of the 20th IEEE International Conference on Software Maintenance (ICSM 2004), pp. 458-462, September 2004.	None, superseded by article below
S. Deelstra, M. Sinnema, J. Bosch, Variability Assessment in Software Product Families, Journal of Information, and Software Technology, conditionally accepted, 2007	All sections in this chapter, except the conclusion. The conclusion was added to link to the next chapter

11.1. Introduction

Before we go into the issues of variability assessment, the first question we need to answer is: why is variability assessment necessary? To answer this question, we go back to the work of Lehman on software evolution. He noted that as the world around us continually changes, the resulting change in purpose and context may render software products useless. It was therefore that Lehman formulated the following law on software evolution: "*A useful software system must undergo continual and timely change or it risks losing market share*" (Lehman et al., 1997). This law applies to all products in a product family.

Although variability in the product family architecture and components anticipates some of the changes in space (different products) and time (different versions of products), not all future changes can be predicted or included in the product family. 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. In the same way that products need to undergo continual change, variability therefore has to undergo continual and

timely change as well, or a product family will risk losing the ability to effectively exploit the similarities of its members.

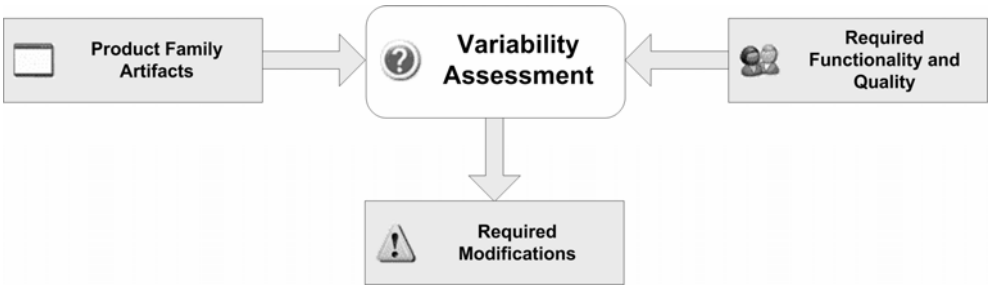

Figure 62. Variability assessment. To determine whether, how, and when variability should evolve, variability assessment evaluates whether the variability in the product family artifacts, matches the variability in the required functionality and quality.

The key challenge in this context is: "in what way can we determine whether, how, and when variability should evolve?". A technique that deals with answering this question is what we refer to as variability assessment. Such a technique answers the question above by analyzing the mismatch between (1) the variability in the product family artifacts and (2) the variability that is demanded as necessary by the differences in functionality and quality in a set of product scenarios (see Figure 62). We call the first type of variability, *provided variability*, and the second type, *required variability*. We call the mismatch between them a *variability mismatch*.

That variability assessment is indeed relevant becomes clear from examining five common activities for software product families in which the 'whether-how-when'-question appears:

Determine the ability of the product family to support a new product: The decision to add a new product to the portfolio depends on how well the new product fits into the product family scope. This fit depends on to which extent the required combinations of features in the new product are supported by the provided variability of the product family.

During product derivation, determine whether mismatches should be implemented in product specific artifacts or integrated in the product family: As product families are focused around a reuse infrastructure, changes can be applied product specifically, or to the reusable product family artifacts. Solving a variability mismatch by changing the reusable product family artifacts is beneficial if the short term and potentially more expensive investment in comparison to product specific adaptation, is outweighed by a decrease in cost of development effort in other products (e.g. due to a decrease in the number and severity of variability

mismatches in the future). Solving a variability mismatch in the product family furthermore depends on whether it is possible or desirable to apply changes to the product family artifacts. A solution may, for example, change dependencies in such a way that dependency values for existing products cannot be met anymore. Variability assessment in this context therefore involves assessing which combinations of features of a new product are not supported by the provided variability, and where potential mismatches need to be solved.

Collecting input data for release planning: During the lifecycle of a product family, organizations collect characteristics (such as functionality and quality) that are required and desired for new and existing products. These characteristics are retrieved from, for example, customers, market analysis and technology forecasting. The result of this collection is typically a long list of required and desired characteristics per product. Due to organizational, economical and technical constraints, however, not all of these characteristics can be implemented in the next release of the products or reuse infrastructure. Release planning is therefore concerned with deciding when to release different versions of the products, including the selection of characteristics offered in specific versions as well as decisions concerning the inclusion of these characteristics in the reuse infrastructure. Release planning involves balancing the objectives regarding the organizational (e.g. staff restrictions), economical (e.g. cost and revenue), and technical (e.g. technical feasibility of feature combinations) constraints. The accuracy of the answer to this problem is, amongst others, influenced by the accuracy of the effort estimates for integrating characteristics product specifically and in the product family, as well as the accuracy of determining in which combinations characteristics can be integrated in the product family artifacts. Variability assessment in this context thus involves identifying mismatches as a result of a set of new product releases, as well as determining how mismatches should be solved.

Assess the impact of new features that cross-cut the existing product portfolio. Some organizations develop a product portfolio that consists of a set of products that interact with each other (e.g. an organization providing systems at both server- and client-side). Rather than focusing on adding entire products or product versions, the focus of variability assessment in this context is on assessing the impact of adding one or more features that influence multiple products in the product family.

Determine whether all provided variability is still necessary. Variation points and variants become obsolete when the need to support different alternatives disappears during evolution, or when predictions made during proactive evolution turn out to be incorrect. In Chapter 5, we identified that the existence and lack of removing obsolete variability was one of the underlying causes of complexity, and had a detrimental effect on the efficiency of product derivation. The aim of assessment in

this context is to identify provided variability that is obsolete with respect to the variability required by products that have been or will be developed, and to determine how the product family should respond.

As software product families in industry have been widely adopted and evolve constantly, organizations that employ product families already perform some form of variability assessment. In the two sections below, we discuss the issues with current approaches (both in practice and related work).

11.2. Variability Assessment Issues

Before a new product is derived, for example, a specification of the product functionality and quality is handed to, typically, software architects. The task of these architects is to assess how much effort will be associated in delivering the product with this specific set of functionality and quality. From what we have seen in several case studies that our group has participated over the years (e.g. as described in Chapter 3-5, or the Dacolian case described in the Chapter 4), current approaches that are used to determine whether, when, and how variability should evolve, are associated to a number of *methodological* and *knowledge* issues.

The *methodological* issues refer to the problems associated to the principles and procedures of current approaches.

Unstructured: Variability assessment is often done by architects without explicit methodological guidance. Instead, they employ an informal process based on their own common sense and experience. These informal processes are typically highly unpredictable with respect to their outcome and required effort.

Reactive instead of proactive: Assessments are furthermore often only applied in case of immediate problems or needs. As a consequence, these assessments suffer from time-pressure and lack of availability of experts; both for the assessment process, and for applying solutions.

Generalized instead of optimal decisions: A third issue is that, in some cases, decisions with respect to evolving variability are generalized over a number of features. In Chapter 4, we presented some extreme forms of generalization we found in industry. For example, one business unit would apply all necessary changes product specifically for each release of the product family, while another business unit would incorporate all necessary changes in the reusable product family artifacts. Both cases lead to problems. Where in the first case the full reuse potential of the product family is not utilized (they re-implement similar functionality in each single product), the second case leads to an unnecessary increase of complexity.

Lack of removing obsolete variability: After a while, the purpose of certain variation points and variants may disappear. Functionality specific to some products can become part of the core functionality of all product family members (Bosch et al., 2001), or perceived alternatives may not be needed after all. Assessments, however, usually only consider the necessity and feasibility of including new functionality, but do not evaluate existing variability with respect to its actual use. The result is an abundance of obsolete variation points and variants. They lead to a situation in which the complexity of the product family only increases during evolution, and the predictability and traceability only decreases. In addition, obsolete variation points result in a situation where engineers start to forget about the provided variability. During the case studies we described in Chapter 5, for example, the interviewees indicated the existence of obsolete parameters from which no one knew what they are for, let alone what the optimal value was.

Addressing only one layer of abstraction: Most existing assessment techniques only focus on one layer of abstraction, i.e. either the architecture (e.g. Clements et al. (2001), Folmer et al. (2004)), or its implementation in code (e.g. Bohner (2001) and Kung et al. (1994)). However, variability is a concern that crosscuts all layers of abstraction. In case a detailed list of changes to the product family artifacts (both architecture and components) is required, this issue therefore drives the need for a technique that is able to address all these layers in a uniform fashion.

The *knowledge* issues refer to the problems associated to the information on which decisions in an assessment are based.

Implicit variability: The last methodological issue (addressing only one layer of abstraction) suggests using a variability model that relates variability information across different abstraction layers. In many organizations, however, no complete and explicit model is available that covers all these layers. As the time and effort that is available for an assessment is limited, specifying a complete explicit model is often not an option. Not using a model at all also proofs problematic, however, as it is difficult to keep an overview of all variation points and their relations (see Chapter 5).

Neglecting implementation dependencies: Even if particular options for functionality and quality are independent from a problem space perspective, the design and implementation of a product family can create additional dependencies between them. The consequence of dependencies as a result of implementation is twofold. First, not all combinations of options provided by a product family can be offered in one product without modification. This means that even if all required options are provided by the product family, the required combination of options may not be available. Second, effort estimates for new functionality and quality

cannot be considered independent from other changes, as those may also have implementation dependencies.

Insufficient number of alternative solutions: When a variability mismatch occurs, several solution strategies may exist to address this mismatch. For example, the software architect has to decide whether to solve a mismatch product specifically, in the reuse infrastructure, or not at all. In addition, he or she can choose from different mechanisms to solve the mismatch. The choice for a particular solution depends on a trade-off between pros and cons of the potential solutions. Examples of pros and cons of a solution are: whether it introduces incompatibilities in the asset base due to new dependencies, whether it imposes the use of immature or unstable technology, and how it affects the effort associated with other changes. The issue we address here involves software architects that only consider a very small number of alternatives, rather than carefully looking for the optimal solution (Bosch et al., 2001).

These knowledge issues cause assessments to produce non-optimal and inaccurate results. The consequence is that, during product derivation, unexpected incompatibilities are identified. Chapter 5 explains that these incompatibilities have a profound impact on the total effort and time-to-market for the product at hand.

11.3. Related work

Existing techniques have been suggested for variability assessment. In the discussion on related work below, we relate variability assessment to existing work on product families, and discuss why existing approaches are not suited to address all variability assessment issues we discussed above.

FAST and SEI's Product Line Practice. Weiss and Lai (1999) formulate basic assumptions with respect to product families, from which two are particularly important in the context of this article: "it is possible to predict the changes that are likely to be needed to a system over its lifecycle", and "it is possible to take advantage of these predicted changes". When it comes to actually determining changes to variability, however, the book lacks precision (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". Also in the SEI's Product Line Practices and Patterns book, the evaluation of variability is quoted as an example of an important evaluation. The book, however, does not present a technique to perform these evaluations. Rather, it suggests modifying existing architecture assessment methods to accomplish this goal (pp. 77-83).

Investment analysis. James Whitey (1996) provides an investment analysis approach that focuses on maximizing ROI of product line assets. Robertson and

Ulrich (1998) also evaluate economical aspects of a product family and deal with planning and scoping the product family architecture. DeBaud and Schmid (2003) provide a similar but more general approach. They also claim that product-centric commonality and variability analysis is better than a domain based view, as the latter provides a flawed economic model for making scoping decisions (DeBaud and Schmid, 2003). The approaches, however, are all based on rough estimates, and focus on the question if certain features, assets or products should be part of the product family rather than how required variability should be realized in the product family artifacts.

Assessment. Assessments typically consist 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. Examples of these approaches are ATAM (Clements et al., 2001), ALMA (Bengtsson et al., 2004), and SALUTA (Folmer et al., 2004). These three approaches respectively assess trade-offs between quality attributes, maintainability, and usability in software architectures. The approaches differ with respect to the required information, elicitation and specification of the scenarios. They focus on analyzing the architecture of single systems, rather than being suitable for all layers of abstraction in a product family.

An approach that does address variability, is the approach presented by Wijnstra (2003). The approach presents a high-level discussion on extracting variability information, and, based on this information, evaluates the provided variability with respect to best practices. However, it is restricted to end-user variability, does not provide specific steps for building up a provided variability specification, and is not focused on specific variability needs of the product family members.

Change management. Change management is a process for ensuring that changes to a software system or product family are traceable, carefully planned, and motivated. The change management process is typically a high-level description of how a change should be handled, defines standard deliverables, as well as an organizational structure. Variability assessment neatly fits into the change management process in product families. Where the descriptions in change management process usually do not go further then saying there are steps such as ‘propose change’, or ‘evaluate change’, variability assessment provides the details that are required to actually propose and evaluate changes to the variability.

11.4. Conclusion

The variability assessment issues we identified above, prevent giving a good answer to the question whether, when, and how variability should evolve. As a

response, we have developed the COVAMOF Software Variability Assessment Method (COSVAM). We present this method in the next chapter.

11.5. References

Bengtsson, P.O., Lassing, N., Bosch, J., van Vliet, H., 2004. "Architecture-level Modifiability Analysis (ALMA)", Journal of Systems and Software, Vol. 69(1-2), pp. 129-147.

Bohner, S.A., 1991. Software Change Impact Analysis for Design Evolution, In Proceedings of 8th International Conference on Maintenance and Re-engineering (ICMR'91), Los Alamitos, California, pp. 292-301.

Bosch, J., Florijn, G., Greefhorst, D., Kuusela, J., Obbink, H., Pohl, K., 2001. Variability Issues in Software Product Lines, Proceedings of the Fourth International Workshop on Product Family Engineering (PFE-4), pp. 11–19.

Clements, P., Kazman, R., Klein, M., 2001, Evaluating Software Architectures, Methods and Case Studies, Addison-Wesley, ISBN 0-201-70482-X.

DeBaud, J.M., Schmid, K., 1999. "A systematic approach to derive the scope of software product lines", Proceedings of the 21st Int. Conf. on Software Engineering, California, USA, pp. 34-43.

Folmer, E., Gurp, J., Bosch, J., 2004, "Architecture-Level Usability Assessment", accepted for EHCI.

Kung, D., Gao, J., Hsia, P., Wen, F., Toyoshima, Y., and Chen, C., 1994. Change Impact Identification in Object Oriented Software Maintenance, Proceedings of the International Conference on Software Maintenance (ICSM'94), IEEE CS Press, Los Alamitos, California, pp. 202-211.

Lehman, M.M., Ramil, J.F., Wernick, P.D., Perry, D.E., Turski, W.M., 1997. "Metrics and Laws of Software Evolution - The Nineties View", Proceedings of the Fourth International Software Metrics Symposium, USA.

Robertson, D. Ulrich, K., 1998, "Planning for product platforms", Sloan Mgt. Review, Vol. 39 (4), pp. 19-31.

Weiss, D.M., Lai, C.T.R., 1999, Software Product-Line Engineering: A Family Based Software Development Process, Addison-Wesley, ISBN 0-201-694387.

Whitey, J., 1996. "Investment analysis of software assets for product lines", Technical report CMU/SEI-96-TR-010, Software Engineering Institute.

Wijnstra, J.G., 2003. Evolving a Product Family in a Changing Context, Proceedings of the 5th International Workshop on Software Product-Family Engineering (PFE-5), pp. 111-128.