A Metric Suite to Support Software Product Line Architecture Evaluation

Edson Alves de Oliveira Junior[1], Itana M. S. Gimenes[2], and José Carlos Maldonado[1]

[1] Universidade de São Paulo (USP), São Carlos-SP, Brazil,
{edsonjr,jcmaldon}@icmc.usp.br
[2] Universidade Estadual de Maringá (UEM), Maringá-PR, Brazil,
itana@din.uem.br

Abstract. *This paper presents a metric suite to support software product line architecture (PLA) evaluation. The metric suite was conceived taking into account the variabilities defined on the UML artefacts of the product line based on specific stereotypes. The metrics definition was supported by the SDMetrics tool. An example illustrates the correlation between the metrics and the PL architecture quality attributes for complexity, maintainability and testability. The contribution of this paper is twofold. First and foremost, to provide an alternative of using the metric suite to propose new metrics to perform PLA evaluation based on quality attributes. Second, to allow the comparison of PL configurations to support decision making regarding the PL feasibility in respect to efforts of PL development. An evaluation concerning the PL complexity is presented showing how the metric suite can support PLA evaluations. Moreover, four additional complex class metrics which are specific for PLA evaluation based on quality attributes are proposed. We are currently working on the proposal of an evaluation model and a tool to support PLA evaluation based on the experimental software engineering concepts.*

Key words: Complexity Metrics. Product Line Architecture Evaluation. Product Line Architecture Quality Attributes.

1 Introduction

The software product line (PL) approach aims at the generation of specific products for a given domain based on the reuse of a core asset, which has a set of common features, as well as a set of variable parts, which represents later design decisions [8, 13]. The composition and the configuration of such assets originate specific products. The PL approach is feasible to domains, which have well-defined common features and variation points. The PL adoption has been increasing in the last years since several companies' successful cases are reported in the literature, such as: Philips, Bosch, Nokia and Toshiba [3, 6, 16, 27].

Currently, one of the most important activity of PL management is the evaluation of the PL architecture (PLA) and its components. This evaluation is

XXXIV Conferencia Latinoamericana de Informática (CLEI 2008), pp. 489-498, 2008.

important both for academic and industrial views [18]. Although there are some approaches to PLA evaluation, most of them do not consider UML-based artefacts and their respective variabilities. Moreover, they are mainly focused on the issues of a particular approach. Some papers propose, for instance, metrics for PLA evaluation based on service utilization of PLA components [14]. However, such metrics do not consider the variabilities modeled inside the PLA components. A correlation between PLA metrics, based on variabilities, and PLA quality attributes is not considered in most of the existing literature.

This paper presents a metric suite for PLA evaluation based on UML models. It takes into account annotated variabilities, modeled using specific stereotypes and UML notes placed in use cases, classes and components of a PL [26]. Metrics collected from an excerpt of a PLA, called "Sorting of Elements", are presented. This example illustrates how to analyse the complexity of a PLA based on the collected metrics and the priorization [I1]of some quality attributes.

This paper is organized as follows: section 2 presents the main concepts of PL and PLA evaluation; section 3 presents our metric suite to evaluate PLA; section 4 presents an example of how to compose metrics for measuring the complexity of a PLA; section 5 presents the related works; and finally, section 6 presents the conclusions and future works.

2 Software Product Line and Architecture Concepts

A PL represents a set of systems, which share common features that satisfy the needs of a particular market's or mission's segment [8, 15, 19, 28]. Such set of systems is also called as product family. The family's members are specific products developed in a systematic way from the PL core assets. The approach allows organizations to explore the similarities of their products, thus, increasing the reuse of the assets and consequently reducing their costs and time-to-market [13].

According to Clements and Northrop [8], Heymans and Trigaux [13], and the SEI's Hall of Fame [16], the benefits obtained with the PL adoption can be considered as: **organizational**, better understanding of the domain and the increase of the quality of products, as well the client's reliability; **software engineering**, better domain analyses and the increase of requirements and assets, better control of product's quality, and establishment of reusable standards and documents; and **business**, reduction of the costs with test and maintainance.

The adoption of a PL approach requires long term management, once the benefits are not immediatly obtained. Thus, a wide variety of products should be produced to consolidate the PL adoption and, therefore, provide a return on investiment (ROI) [5].

The Software Engineering Institute (SEI), through its Product Line Practice (PLP) initiative, established the essential activities of a PL approach [17], which are: Core Asset Development (or Domain Engineering), Product Development (or Application Engineering), and Management.

The **Core Asset Development** aims at the development of the main infrastructure that will be used to generate PL products. This activity has as input: product constraints; architectural styles, patterns and frameworks; production strategy; and pre-existing assets repository. The main outputs are: PL scope; PL core asset; and the product production plan. In this activity it is also possible to identify additional requirements that were not previously specified, and, consequently, update the PL core asset. The **Management** activity should insure that the technical activities are performed according to a coordinated planning. One of the most important management activities is the creation of an adoption plan, which describes the desirable state of the organization, and a strategy to reach such state. The PL management should establish how the core asset development and the product development activities can interact to allow the PL evolution. An important activity performed during the PL management is the PLA evaluation.

The PLA plays a central role in the development and evolution of a PL. It is one of the most important artefacts to successfully generate specific products throughout the PL lifecycle. The PLA represents the abstraction of all possible product's architecture that a PL can generate. Organizations that follow the PL approach should continuously evaluate the quality of their products by insuring the quality of their PLA. So, the PLA evaluation should be considered one of the most important activity throughout a PL lifecycle [8].

The evaluation of a PLA requires a set of metrics [9], that can evidence the PL quality and serves as the basis to analyze the managerial and economic values of a PL [5]. The PLA is different from the architecture of a single product as it has to explicitly model product similarities and variabilities. The variability management is an important issue of PL management [4, 12, 26]. It begins in the requirements analyses and affects most of the PL assets. The analyses of variabilities in the product development can determine the aggregated value of a PL for an organization. So, the PLA evaluation should consider the following aspects [11]: relevant quality attributes of a PLA; time when the PLA is evaluated; and techniques and metrics used to evaluate the PLA.

Oliveira Junior et al. [25] present the state of the art on PLA evaluation. Current works were grouped according to their main goals, as follows: **(i) evaluation of PLA quality attributes -** works that focus on the use of scenarios, GQM (Goal Question Metric) approach, ATAM (Architecture Tradeoff Analysis Method) and SAAM (Software Architecture Analysis Method) methods and ADLs (Architecture Description Language); **(ii) PLA structural evaluation -** works that focus on the use of metrics to evaluate components; and **(iii) definition and evaluation of PL scoping -** works that focus on the use of cost and economic models.

Taking the concepts presented in this section as a basis, the next section presents the proposed metric suite to support PLA evaluation, regarding quality attributes evaluation, as well structural evaluation of PLAs.

3 A Metric Suite to Evaluate PLA

The metric suite was conceived based on PL variabilities, which are composed of variation points and variants annotated in UML use cases, classes and components by using stereotypes [26]. A variability allows selection from different options, called variants [28]. Each variability, represented by a UML note, may contains one or more variation points. A variantion point, represented by the stereotype << *variationPoint* >>, has a set of associated variants. The relationship between a variation point and variants can be one of the following:

- **alternative inclusive (OR):** represented by the << *alternative_OR* >> stereotype. One or more variants can be chosen;
- **alternative exclusive (XOR):** represented by the << *alternative_XOR* >> stereotype. Only one variant is chosen;
- **optional:** represented by the << *optional* >> stereotype. The variant can be or not present in a product; and
- **mandatory:** represented by the << *mandatory* >> stereotype. The variant is always present in a product.

Figure 1 illustrates the use of some of these stereotypes in a class diagram.

The metric suite is composed of 35 metrics, which are presented in Table 1. The proposed metrics were defined using the SDMetrics tool [30], which allows the collection of metrics based on UML metamodels. The metrics are divided according to the UML element they measure, which are: use cases, classes, components, diagrams, and UML models representing the overall PL.

Next section presents an example of the use of the proposed metric suite aimed at composing more complex metrics to evaluate PLA regarding the following quality attributes: complexity, maintainability and testability.

4 Measuring the PLA's Complexity

This section shows how to measure the complexity of a variability for an algorithm of "Sorting of Elements". Four more complex metrics regarding the quality attributes for maintainability and testability were defined: *CompVariant*, *CompVP*, *CompVariability* and *CompPL*. These metrics are composed from the metric suite. The example shows three different product configurations, from a PL core asset. As a result it points out at the less complex product configuration of the PL. The analysis of the quality attributes has as a basis the McCabe's [21] Cyclomatic Complexity (CC) and Weighted Methods per Class (WMC) metrics. The CC is the measure of lineary independent paths of a source code, and the WMC is the sum of the CC of all methods for a class. The four composed metrics defined to measure complexity of a PLA are as follows:

CompVariant is the complexity measure (WMC value) for a given class, defined as:

$CompVariant = WMC$

Table 1. Proposed Metrics to Support PLA Evaluation.

Ord.	Metric	Description
01	UseCaseVP	Use case is a variation point.
02	UseCaseAlternativeOR	Use case is an alternative inclusive variant.
03	UseCaseAlternativeXOR	Use case is an alternative exclusive variant.
04	UseCaseOptional	Use case is an optional variant.
05	UseCaseMandatory	Use case is a mandatory variant.
06	UseCaseNumVariantsAltOR	Number of use case alternative inclusive variants.
07	UseCaseNumVariantsAltXOR	Number of use case alternative exclusive variants.
08	UseCaseNumVariantsOptional	Number of use case optional variants.
09	UseCaseNumVariantsMandatory	Number of use case mandatory variants.
10	ClassVP	Class is a variation point.
11	ClassAlternativeOR	Class is an alternative inclusive variant.
12	ClassAlternativeXOR	Class is an alternative exclusive variant.
13	ClassOptional	Class is an optional variant.
14	ClassMandatory	Class is a mandatory variant.
15	ClassNumVariantsAltOR	Number of class alternative inclusive variants.
16	ClassNumVariantsAltXOR	Number of class alternative exclusive variants.
17	ClassNumVariantsOptional	Number of class optional variants.
18	ClassNumVariantsMandatory	Number of class mandatory variants.
19	UseCaseTotalVP	Number of variation points in a use case diagram.
20	UseCaseTotalAlternativeOR	Number of altern. incl. variants in a use case diagram.
21	UseCaseTotalAlternativeXOR	Number of altern. excl. variants in a use case diagram.
22	UseCaseTotalOptional	Number of optional variants in a use case diagram.
23	UseCaseTotalMandatory	Number of mandatory variants in a use case diagram.
24	UseCaseTotalVariabilities	Number of variabilities in a use case diagram.
25	ClassTotalVP	Number of variation points in a class diagram.
26	ClassTotalAlternativeOR	Number of altern. incl. variants in a class diagram.
27	ClassTotalAlternativeXOR	Number of altern. excl. variants in a class diagram.
28	ClassTotalOptional	Number of optional variants in a class diagram.
29	ClassTotalMandatory	Number of mandatory variants in a class diagram.
30	ClassTotalVariabilities	Number of variabilities in a class diagram.
31	ComponentTotalVariabilities	Number of variabilities in a component diagram.
32	UseCaseTotalPLVariabilities	Number of variabilities in use cases of a PL.
33	ClassTotalPLVariabilities	Number of variabilities in classes of a PL.
34	PLTotalVariability	Number of variabilities in a PL.
35	ComponentVariable	Component has variation.

CompVP is the complexity measure for a given class, that is a variation point, plus the sum of the complexity of each associated variant of such variation point. This metric is defined as:

$$CompVP = CompVariantVP + \sum_{i=1}^{nVariants} CompVariant_i \text{ , where:}$$

$CompVariantVP = CompVariant$ value of the class that is a variation point.

$nVariants = ClassNumVariantsAltOR + ClassNumVariantsAltXOR + ClassNumVariantsOptional + ClassNumVariantsMandatory$ (metrics from section 3).

CompVariability is the sum of the measure ***CompVP*** of each variation point of a given variability, defined as:

$CompVariability = \sum_{i=1}^{nVP} CompVP_i$, where:

nVP = number of variation points of a given variability

CompPL is the sum of the measure ***CompVariability*** of each variability for a given PL, defined as:

$$CompPL = \sum_{i=1}^{PLTotalVariability} CompVariability_i$$

To illustrate the collection of the metrics defined above, suppose that we have a PL and it has a set of features, like the "Sorting of Elements". This feature may have a variability associated. This variability may have in turn two variation points, which are: the kind of the sorting algorithm, and the kind of element to be sorted. Figure 1 presents the class diagram for the variability "Sorting of Elements".

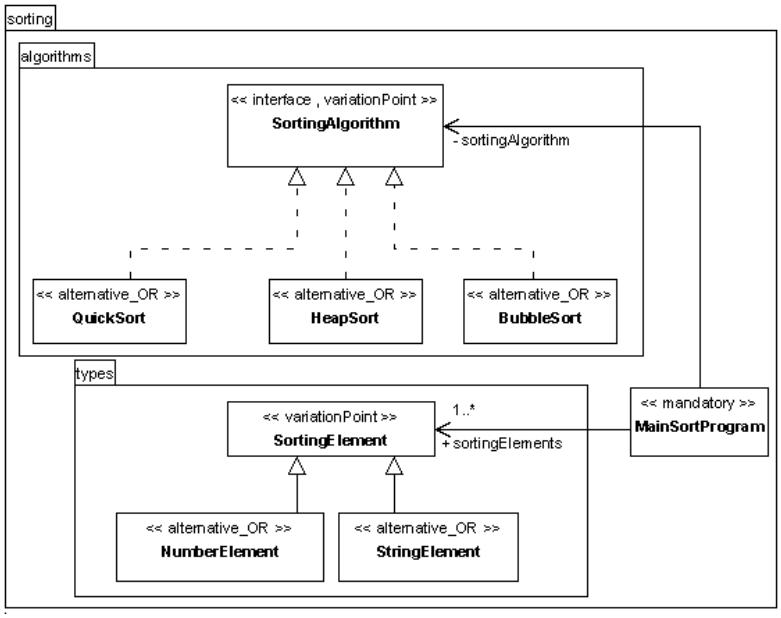

Fig. 1. Class Diagram for the Variability "Sorting of Elements".

The variation points (*SortingElement* and *SortingAlgorithm*) are annotated with the stereotype << *variationPoint* >>. The variants are annotated with

the stereotypes << *alternative_OR* >> (*NumberElement*, *StringElement*, *QuickSort*, *HeapSort* and *BubbleSort*) and << *mandatory* >> (*MainSortProgram*).

The metric *CompVariant* was collected for each class and interface, using the Metrics plugin for Eclipse [22], as follows: *MainSortProgram* = 03; *SortingAlgorithm* = 00 (interface); *QuickSort* = 11; *HeapSort* = 13; *BubbleSort* = 08; *SortingElement* = 01; *StringElement* = 04; and *NumberElement* = 04.

Thus, the metrics *CompVP*, *CompVariability* and *CompPL* were calculated analysing three possible products configurations based on the combination of variants for each product from the variability "Sorting of Elements".

Product configuration 1, composed of *MainSortProgram*, *SortingElement*, *NumberElement*, *QuickSort*, *BubbleSort* and *SortingAlgorithm*, obtained the following values:

- $CompVP_{SortingAlgorithm}$ = *SortingAlgorithm* (00) + *QuickSort* (11) + *BubbleSort* (08) = 19
- $CompVP_{SortingElement}$ = *SortingElement* (01) + *NumberElement* (04) = 05
- $CompVariability = CompVP_{SortingAlgorithm}$ (19) + $CompVP_{SortingElement}$ (05) + *MainSortProgram* (03) = 27
- $CompPL = CompVariability$ = 27

Product configuration 2, composed of *MainSortProgram*, *SortingElement*, *NumberElement*, *QuickSort*, *HeapSort* and *SortingAlgorithm*, obtained the following values:

- $CompVP_{SortingAlgorithm}$ = *SortingAlgorithm* (00) + *QuickSort* (11) + *HeapSort* (13) = 24
- $CompVP_{SortingElement}$ = *SortingElement* (01) + *NumberElement* (04) = 05
- $CompVariability = CompVP_{SortingAlgorithm}$ (24) + $CompVP_{SortingElement}$ (05) + *MainSortProgram* (03) = 32
- $CompPL = CompVariability$ = 32

Product configuration 3, composed of *MainSortProgram*, *SortingElement*, *StringElement*, *HeapSort*, *BubbleSort* and *SortingAlgorithm*, obtained the following values:

- $CompVP_{SortingAlgorithm}$ = *SortingAlgorithm* (00) + *HeapSort* (13) + *BubbleSort* (08) = 21
- $CompVP_{SortingElement}$ = *SortingElement* (01) + *StringElement* (04) = 05
- $CompVariability = CompVP_{SortingAlgorithm}$ (21) + $CompVP_{SortingElement}$ (05) + *MainSortProgram* (03) = 29
- $CompPL = CompVariability$ = 29

Now, we can find out that the complexity relation between the products is:

$$\boxed{CompPL_{configuration1} < CompPL_{configuration3} < CompPL_{configuration2}}$$

The results show that the first product has less complexity than the other ones, being the best case to resolve the variability "Sorting of Elements". The

second product is the worst case to resolve such variability. So, the generation of products similar to the first one is more simple than the generation of similar products to the second and third ones. So, we can say the same for the third product regarding the second one.

5 Related Work

Lopez-Herrejon and Trujillo [20] propose complexity analyses of PLs based on the McCabe's Cyclomatic Complexity metric [21], as we do. However, they consider the PL variability modeled using XVCL (XML-based Variant Configuration Language) instead of using UML models. Moreover, they do not consider, as we do in section 4, the combination of the variability, variation points and variants relationships in order to analyse the more feasible PL configuration regarding its complexity.

Van der Hoek et. al [14] propose metrics for components to analyse structural problems, which are Required Service Utilization (RSU) and Provided Service Utilization (PSU). However, such metrics are based on the concept of service that is defined as any public accessible resource in an architecture described using an Architecture Description Language (ADL). We do not consider PLA descriptions in ADL, but in UML models.

Ajila and Dumitrescu [1] measure the PL evolution in terms of Lines of Code (LOC), not at the variation point view as we do.

Aldekoa et al. [2] extended the Maintainability Index to emcompass features and apply it to a simple case study. The metric is calculated based on the average of the McCabe's Cyclomatic Complexity value. However, it is based on the generated code and not on the PL modeled variability.

Rahman [29] proposes a component structural metric suite to measure PLA quality attributes like observability, configurability, interface complexity, modularity, service utilization, and maturity. A set of metrics is defined to evaluate each PLA quality attribute. Although Rahman's set of metrics is not based on the variabilities modeled in UML, it can contribute to better understand how to evaluate those quality attributes regarding variabilities in UML models.

6 Conclusions and Future Work

Current works propose metrics that provide significant aggregated value to support design decisions in PL development. However, such works do not consider variabilities modeled on the PL artifacts. Our metric suite was defined based on variabilities modeled on the UML artifacts of the PL such as use cases, classes and components. Composed metrics can be calculated from the metric suite to evaluate PLA based on the prioritization of quality attributes. An example of such metric support is illustrated in this paper by defining a set of four metrics to evaluate PLA, at class level, based on the following quality attributes: complexity, maintainability and testability. Thus, we provided means to compare

configurations of a given PL and to indicate which configuration is more feasible based on the measured complexity.

We are currently working on the proposal of a PLA evaluation method, which is composed of: (i) a metric suite; (ii) an evaluation model; (iii) a tool to support the definition and execution of formal experiments based on the collected metrics and the evaluation model. The evaluation method aimed at answering managerial issues, such as: (i) What is the complexity of a specific variability in a PL? (ii) What is the impact of include/update/remove a specific feature of a PL? (iii) What is the less complex configuration of a PL? We are currently investigating the possibility of extending use case points [7, 23] to emcompass PL variabilites. Moreover, we are considering to provide the definition of formal experiments applying the proposed metrics in a PL for Workflow Management Systems.

Acknowledgments. The authors would like to thank Jürgen Wüst for providing a full-featured SDMetrics release, which allowed us to define our PLA evaluation metrics.

References

1. Ajila, S., Dumitrescu, R. T.: Experimental Use of Code Delta, Code Churn, and Rate of Change to Understand Software Product Line Evolution. Journal of Systems and Software. 80(1), 74-91 (2007)
2. Aldekoa, G., Trujillo, S., Sagardui, G., Díaz, O.: Experience Measuring Maintainability in Software Product Lines. In: Proceedings of the XI Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2006), pp. 173-182. (2006)
3. Bass, L., Clements, P., and Kazman, R.: Software Architecture in Practice. Addison-Wesley, Boston (2005)
4. Becker, M.: Towards a General Model of Variability in Product Families. In: Proceedings of the Software Variability Management Workshop, pp. 19-27. (2003)
5. Bockle, G., Clements, P., McGregor, J. D., Muthig, D., and Schimid, K.: Calculating ROI for Software Product Lines. IEEE Software. 21, 23-31 (2004)
6. Buhrdorf, R., Churchett, D., and Krueger, C. W.: Salion's Experience with a Reactive Software Product Line Approach. Lecture Notes in Computer Science. 3014, 317–322 (2003)
7. Carroll, E. R.: Estimating Software Based on Use Case Points. In: Proceedings of the 20th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, pp. 257-265. (2005)
8. Clements, P. and Northrop, L.: Software Product Lines: Practices and Patterns. Addison-Wesley Longman Publishing Co., Boston (2001)
9. Dincel, E., Medvidovic, N., and van der Hoek, A.: Measuring Product Line Architectures. In: Proceedings of the Fourth International Workshop on Product Family Engineering, pp. 346-352. (2001)
10. Dobrica, L. and Niemelä, E.: A Survey on Software Architecture Analysis Methods. IEEE Transactions on Software Engineering. 28, 638-653 (2002)
11. Etxeberria, L. and Sagardui, G.: Product-Line Architecture: New Issues for Evaluation. In: Proceedings of the Internation Software Product Line Conference (SPLC 2005), pp. 174185 (2005)

12. Gurp, J. V., Bosch, J., and Svahnberg, M.: On the Notion of Variability in Software Product Lines. In: Proceedings of the Working IEEE/IFIP Conference on Software Architecture (WICSA01), IEEE Computer Society (2001)
13. Heymans, P. and Trigaux, J. C.: Software Product Line: State of the Art. Technical report, PLENTY project, Institut dÍnformatique FUNDP (2003)
14. Hoek, A.; Dincel, E.; Medvidovic, N.: Using Service Utilization Metrics to Assess the Structure of Product Line Architectures. In: Proceedings of the 9th International Symposium on Software Metrics (METRICS 2003), pp. 298. IEEE Computer Society, Washington, DC, USA (2003)
15. Northrop, L. M.: Sei's Software Product Line Tenets. IEEE Software. 19, 32-40 (2002)
16. SEI's Hall of Fame, `http://www.sei.cmu.edu/productlines/plp_hof.html`
17. SEI - A Framework for Software Product Line Practice, `http://www.sei.cmu.edu/productlines/framework.html`
18. Linden, F. van der, Bosch, J., Kamsties, E., Känsälä, K., Krzanik, L., and Obbink, J. H.: Software Product Family Evaluation. In: Proceedings of the Software Product-Family Engineering (PFE 2003), pp. 352-369. (2003)
19. Linden, F. J. van der, Schmid, K., Rommes, E.: Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering. Springer (2007)
20. Lopez-Herrejon, R. E., Trujillo, S.: How complex is my Product Line? The case for Variation Point Metrics. In: Proceedings of the 2nd International Workshop on Variability Modelling of Software-Intensive Systems (VAMOS 2008), pp. 17-23. Essen, Germany (2008)
21. McCabe, T. J.: A Complexity Measure. In: Proceedings of the 2nd International Conference on Software Engineering (ICSE 1976), IEEE Computer Society Press, USA, pp. 407. (1976)
22. Metrics 1.3.6 - Plugin for Eclipse IDE. `http://metrics.sourceforge.net`
23. Mohagheghi, P., Anda, B., Conradi, R.: Effort Estimation of Use Cases for Incremental Large-Scale Software Development. In: Proceedings of the 27th International Conference on Software Engineering (ICSE 2005), pp. 303-311. (2005)
24. Muccini, H. and van der Hoek, A.:Towards Testing Product Line Architectures. Electronic Notes on Theor. Computer Science 82, (2003)
25. Oliveira Junior, E. A., Maldonado, J. C., and Gimenes, I. M. S.: Uma Revisão Sistemática sobre Avaliação de Linha de Produto de Software. Technical report, Instituto de Ciências Matemáticas e de Computação (ICMC) - Universidade de São Paulo (USP) (2007)
26. Oliveira Junior, E. A., Gimenes, I. M. S., Huzita, E. H. M., and Maldonado, J. C.: A Variability Management Process for Software Product Lines. In: Proceedings of the 2005 Conference of the Centre for Advanced Studies on Collaborative Research (CASCON 2005), pp. 225-241. IBM Press (2005)
27. Ommering, R. van: Building Product Populations with Software Components. In: Proceedings of the 24th International Conference on Software Engineering (ICSE 2002), pp. 255-265. ACM, New York (2002)
28. Pohl, K., Böckle, G., Linden, F. J. van der: Software Product Line Engineering: Foundations, Principles and Techniques. Springer-Verlag, New York (2005)
29. Rahman, A.: Metrics for the Structural Assessment of Product Line Architecture. (2004)
30. SDMetrics - The UML Design Quality Metrics Tool, `http://www.sdmetrics.com`