Towards a Crosscutting Approach for Variability Management

Rodrigo Bonifácio and Paulo Borba
Informatics Center
Federal University of Pernambuco
Recife, Brazil
{rba2, phmb}@cin.ufpe.br

Abstract

Variability management is a common challenge for Software Product Line (SPL) adoption, since developers need suitable mechanisms for describing or implementing variability that might occur at different SPL views (requirements, design, implementation, and test). In this research, we are proposing a novel approach for variability management. The main goal is to improve the separation of concerns between languages used to manage variabilities and languages used to specify software engineering assets. Initial results gave evidence that, by applying our proposed approach, both representations can be understood and evolved in a separated way.

1. Introduction

In order to reduce time-to-marketing and improve quality of software products, several approaches and techniques for economy of scope, mass customization, and systematic reuse have been proposed. Examples of such approaches include *Software Product Lines* (SPLs) [18, 8], *Generative Programming Techniques* [9], and *Software Factories* [13]. Actually, there are commonalities among such approaches. For instance, one common characteristic is the relevance of domain analysis, which aims at defining a scope (often in the business sense) in which reusable assets can be used for generating specific products.

Additionally, it is a common practice to use *feature modeling* for representing features that are common to all products within a scope and which features are optional, being useful for differentiating specific products in a family. Therefore, aiming to generate specific products, it would be necessary to: (a) introduce suitable variation points in common assets, (b) develop variant assets that extend these variation points, and (c) relate features to both common and variant assets. In this work, we consider *variability management* as the discipline that guides these activities. Actually, variability management is an interesting kind of crosscutting concern, since certain features require variation points to be spread in different places of requirements, design, code, and test artifacts. This crosscutting nature of variability management results in interesting challenges regarded to SPL traceability, evolvability, and product derivation.

As a consequence, several authors have proposed the use of *aspect-oriented* techniques to better modularize the composition of common and variant assets of a product line [11, 1, 2]. In this thesis we go beyond this composition issue. We mainly consider a more encompassing notion of variability management, presenting its semantics as a crosscutting concern and describing the contribution of relevant artifacts (such as feature models and configuration knowledge) in product generation.

Our hypotheses is that a clear separation between variability management and common software engineering artifacts reduces the effort for evolving a SPL and for reasoning about feature traceability. A clear separation means that each SPL model (feature model, configuration knowledge, SPL use case model, and so on) should focus on specific concerns. For instance, use case models should represent just the valid interactions between external actors and the system. They should not be enriched to describe variability space

(as proposed in [4]). The challenge is that, in order to generate specific products, the clear separation that we are proposing requires composition processes involving different SPL models. In this thesis, we define the semantics of composition processes as crosscutting mechanisms. The elegant notion of crosscutting mechanisms, formalized by Masuhara and Kiczales [16], is used as underlining support for presenting the semantics of our composition processes.

Initial results of our approach, applied in the context of representing SPL variabilities in use case scenarios, revealed to us improvements in both evolvability and product generation [6]. The choice of applying our approach in this context was motivated because current techniques for scenario variability management [10, 4] do not present a clear separation between variability management and scenario specification. In summary, the contributions of this thesis are threefold

- Characterize the broader notation of variability management as a crosscutting concern and, in this way, propose an approach for representing it as an independent view of the SPL.
- A framework for modeling the composition processes of variability mechanisms. This framework gives a basis for describing variability mechanisms (such as scenario composition and parameterization), allowing a better understanding of each mechanism and highlighting the contribution of each model used in the composition processes.
- A deeper evaluation of existing techniques for representing scenario variability. Such an evaluation will take into consideration not only the support for different variability techniques (parameterization, optional scenarios), but also a comparison of existing works with regard to SPL evolvability and traceability.

The next section describes our approach, named as *variability management as crosscutting* (Section 2). It also relates some open questions of our approach, which we are going to solve in this thesis. After that, Section 3 presents the results of three empirical studies, which we have compared our proposed approach with existing works. These comparisons were based on *Design Structure Matrices* and on a suite of metrics, adapted from aspect-oriented community, for quantifying modularity. We also present in Section 3 a discussion about several improvements on our evaluation processes. Then, we relate our thesis with existing works (Section 4) and present final concludings in Section 5.

2 Variability management as crosscutting

In order to represent variability management as a crosscutting concern, we proposed a modeling framework (Section 2.1), that slightly generalizes the Masuhara and Kiczales (MK) framework [16], and instantiate it for the product line domain. The MK framework aims to explain how different *aspect-oriented* technologies support crosscutting modularity. In their proposed approach, each technology is modeled as a three-part description: the related weaving processes take two programs as input, which crosscut each other with respect to the resulting program or computation [16].

Specifically in the context of use case scenario variabilities, we represent the semantics of **scenario variability management** as a weaver that takes as input four specifications (*product line use case model*, *feature model*, *product configuration*, and *configuration knowledge*) that crosscut each other with respect to the resulting product specific use case model (Figure 1). Combining these input languages, it is possible to represent the kinds of variability that we are interested in: *optional use cases and scenarios*, *quantified changed scenarios*, and *parameterized* scenarios.

In what follows, we present our modeling framework, proposed to explain variability management as a crosscutting concern. Then, we instantiate such a framework (Section 2.2) for representing one scenario variability technique — optional use cases and scenarios. More details about our approach can be found elsewhere [5].

2.1 Modeling framework

As explained before, our modeling framework, proposed for representing variability management weaving processes, is based on the Masuhara and Kiczales work [16].

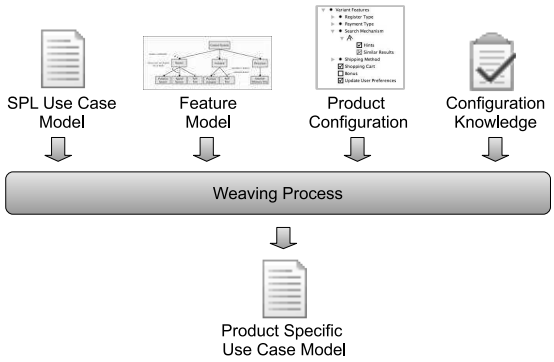

Figure 1. Overview of our weaving process.

Thereby, we describe each variability technique as a weaver. In our work, we represent these weavers as an *6-tuple* (see Eq. 1 and Table 1), in such a way that we can describe the contribution of each input model used in the composition process (Figure 1).

$$Weaver = \{o, o_{jp}, L, L_{id}, L_{eff}, L_{mod}\}, \quad (1)$$

Table 1. Modeling framework elements.

Element	Description
o	Output language used for describing the results of the weaving process
o_{jp}	Set of join points in the output language
L	Set of languages used for describing the input specifications
$L_{ID}(l)$	Set of constructions in each input language l, used for identifying the output join points
$L_{EFF}(l)$	For each input language l, this element represent the effect of its constructions in the weaving process
$L_{MOD}(l)$	Set of modular unities of each input language l

As a consequence, we model each variation technique by filling in the six parameters of our *6-tuple* representation. In order to do that, we first provide a reference implementation for the corresponding weaver. After that, we state how elements of the reference implementation correspond to elements of the modeling framework. In this way, we can represent the semantics of a variability technique at the same time that we explain the contribution of each involved model. Altogether, by applying our approach, it is possible to design variation techniques that present a better separation between variability models and common software engineering artifacts.

In our case studies, we have represented the reference implementations (and the meta-model of the input and output models) using the Haskell programming language. This leads to concise weaving processes descriptions and keeps our model close to MK work, where weaving processes are specified in the Scheme programming language.

2.2 Modeling framework instance

In this section we present an instance of our modeling framework. This instance describes the variation technique responsible for selecting, based on a specific product configuration, which scenarios will be assembled in a product. Notice that we do not present the semantics of other scenario variability techniques in this paper. More examples can be found elsewhere [5].

At first, consider a SPL (*eShop product line*) for the electronic commerce domain, whose part of feature model is depicted in Figure 2. Additionally, consider the rules for product derivation described in Table 2. Actually, such a table represents our configuration knowledge, being responsible for relating feature expressions to artifacts.

In this example, the configuration knowledge enforces that any product that includes *Shopping Cart* and *Bonus* features will be configured with the optional scenario *Buy Products with Cart*. In a similar way, any product that includes *Update User Preferences* feature will be assembled with the optional scenario *Register User Preference*. In our approach, scenarios are obliviousness about being mandatory or optional. The configuration knowledge is responsible for enforcing such design decisions.

Listing 1 presents the product derivation function (*pdWeaver*), which corresponds to the reference implementation for the *optional use cases and scenarios* technique. This function takes as input a *SPL use case model* (UCM), a *feature model* (FM), a *product configuration* (PC), and a *configuration knowledge* (CK).

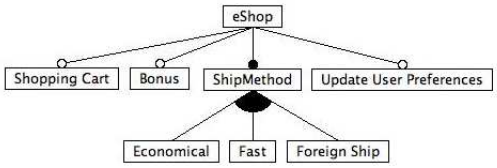

Figure 2. Subset of eShop feature model.

Table 2. eShop configuration knowledge

Expression	Required Scenarios
eShop	Proceed to Purchase
	Search for Products
	...
not (Cart **and** Bonus)	Buy a Product
Cart **and** Bonus	Buy Products with Cart
Update Preferences	Register User Preferences
...	...

Initially, this function verifies if the product configuration is a well formed instance of the feature model (Line 3 in Listing 1) — if it is not the case, an *InvalidProduct* error is thrown. Then, the IDs of selected scenarios are filtered by the *configure* function. This is done by evaluating which feature expressions, defined in the list elements (*x:xs*) of the configuration knowledge, are valid for the specific product instance (*eval* function). Finally, given the resulting list of scenario IDs, the function *retrieveScenarios* returns the product specific scenarios.

Listing 1. Product derivation weaver function

```
1  pdWeaver :: UCM -> FM -> PC -> CK -> ScenarioList
2  pdWeaver ucm fm pc ck =
3   if not (validInstance fm pc)
4    then error InvalidProduct
5    else retrieveScenarios ucm (configure pc ck)
6
7  configure :: PC -> CK -> ListOfScenarioId
8  configure pc (CK []) = []
9  configure pc (CK (x:xs)) =
10  if (eval pc (expression x))
11   then (artifacts x) ++ (configure pc (CK xs))
12   else configure pc (CK xs)
```

It is important to notice that this variation technique presents two levels of crosscutting. First, the feature model, the product configuration, and the configuration knowledge crosscut each other with respect to the list of valid scenario IDs. Then, the resulting list of scenario IDs crosscuts with the use case model for selecting the product specific scenarios.

The model of the *optional use cases and scenarios* technique, in terms of our modeling framework, is shown in Table 3. The *pdWeaver* function is used to argue that the model is realizable and appropriate. We achieve this by matching the model elements to corresponding parameters and auxiliary functions in the reference implementation. Therefore, the input models UCM, FM, CK, and PC are represented as different parameters of the *pdWeaver* function. An instance of the UCM corresponds to the specification of all SPL scenarios. A FM instance is only responsible for declaring the SPL features and the relationships between them. As a consequence, there is no coupling between FMs and UCMs. On the other hand, relationships between features and artifacts are documented in the configuration knowledge. Finally, the PC specifies which features were selected for a specific product. We present a discussion about coupling in Section 3.

The UCM has a greater importance over the other input languages (UCM_{EFF}), since it declares the product specific scenarios (the output of this weaver process generated by the *pdWeaver* function). These scenarios (UCM_{ID}) are used in the *retrieveScenarios* function in order to identify which artifacts will be assembled in the final product.

In order to identify which artifacts are required for a specific product, the *configure* function (CK_{EFF}) checks the feature expression (CK_{ID}) against the product specific features (PC_{ID}). The effect of FM in this weaver (FM_{EFF}) is to check if the PC is well formed. Such evaluation is implemented by the *validInstance* function and considers the PC feature selection (PC_{EFF}).

As mentioned before, it is beyond the scope of this paper presenting other instances of our modeling framework, although we have already instantiated it for two other scenario variability techniques: *quantified changed scenarios* and *parameterized scenarios*. Next, we point some activities that we have planed to evolve our variability model.

Table 3. Model of Product Derivation

Element	Description
o	Product specific scenarios (list of scenarios)
o_{jp}	Scenario declarations
L	{UCM, FM, CK, PC}
UCM_{ID}	SPL scenarios
FM_{ID}	SPL features
CK_{ID}	Feature expressions and scenario IDs
PC_{ID}	Product specific feature selection
UCM_{EFF}	Provides declaration of scenarios
FM_{EFF}	Checks if a SPL instance is well formed
CK_{EFF}	Identifies selected artifacts
PC_{EFF}	Triggers scenario selection
UCM_{MOD}	Scenario
FM_{MOD}	Feature
CK_{MOD}	Each pair $(expression, artifact\ list)$
PC_{MOD}	Feature

2.3 Modeling framework improvements

Previous results of our work reveal to us research activities that can be conduced in the scope of our modeling framework. Below we summarize part of these activities.

- **Improve type checking.** The current version of our variability framework is structured in a single phase, responsible for both type checking and for weaving the input models. Recently, we have realized the importance of checking properties of the input models before the composition process. These properties correspond to *preconditions* that should hold, in order to verify the correctness of a composition. Therefore, this activity aims to identify other properties that should be checked and restructure our variability framework to modularize type checking and composition processes in different phases.

- **Improve the configuration knowledge.** The variability techniques that we have represented [5] are well supported by a model of configuration knowledge that relates feature expressions to artifacts. If a feature expression is valid for a specific instance, the related artifacts will be assembled in the final product. However, we have identified other kinds of scenario variability that can be easily represented using a more expressive view of the configuration knowledge. In this view, feature expressions are related to *model transformations*, instead of being related to software artifacts. If a feature expression is valid for a specific instance, the *model transformation* will be applied. However, we have not evaluated this design yet.

- **Apply our model to other contexts.** Until now, we have applied our variability framework for the context of use case scenarios. As a future work, we aim at representing, using our approach, techniques for SPL test case variability. By doing that, we will be able to identify, more precisely, the real benefits regarded to SPL traceability.

The next section presents some results that we have achieved based on evaluating our approach. Additionally, it also presents future activities for improving the evaluation process.

3 Evaluation

In a previous paper [6], we reported on the benefits of a clear separation between variability management and use case scenarios. We achieved such a result by comparing our approach with PLUC [4] and PLUSS [10], two representative notations for SPL scenario variability. In order to increase our confidence, different techniques for evaluating modularity had been used: *Design Structure Matrices* (Section 3.1), quantitative analysis associated to a proposed metric suite (Section 3.2), and observations of the effort needed to introduce SPL increments using each approach (not detailed in this paper).

In this section, we present just the evaluation techniques. Detailed results of our analysis can be found elsewhere [6, 5]. Additionally, Section 3.3 presents how we are planning to evolve our evaluation process.

3.1 Design Structure Matrices (DSMs)

DSMs is an interesting and simple tool for visualizing dependencies between design decisions [3]. Such

decisions appear in the rows and columns of a matrix. We can identify a dependency by observing the columns in a given row [3]. For example, the first row in Figure 3 indicates that design decisions regarded to variability space (feature model) depends on the task of creating the use case models in the PLUC approach [4]. The first can not be independently performed after the second. This occurs because variability space is specified at specific sections of use cases scenarios in PLUC. Therefore, it is not possible to evolve variability management (introducing new features, products, or relations between features and artifacts) without reviewing the use case model. This is expressed in the non modular DSM of Figure 3, which depicts cyclical dependencies between use cases and variability management artifacts.

		1	2	3	4
1	Feature model		x		
2	Use case model	x		x	x
3	Product configurations	x	x		
4	Configuration knowledge	x	x		

Figure 3. DSM Analysis of PLUC

Our approach reduces the dependencies between variability management and scenario specifications (Figure 4). For instance, changes in feature model or new definitions of products do not require changes in the use case model.

		1	2	3	4	5
1	Feature model					
2	Mapping	x				
3	Use case model		x			
4	SPL instances	x				
5	Configuration model	x		x		

Figure 4. DSM Analysis of our approach

We have applied DSMs to visualize design dependencies in two levels [6]: the first one presents a high level view of dependencies between variability management (feature model, SPL instances and configuration model) and use cases (as shown in Figure 3 and Figure 4); the second one presents how features are spread among use cases. Although we do not present any example of the second level of DSM in this paper, it is very useful for computing several of the metrics present in the next section.

3.2 Proposed metric suite

We derived several metrics for quantifying feature modularity and use case model complexity (related to the size of specifications) [6]. Increasing levels of feature modularity implies better evolvability, since SPL changes or increments can be performed in a isolated way. Also, if a modular feature specification could crosscut other specifications, it would be expected more reusable assets. The metric suite described in this section was adapted from [12] for both product line and use cases contexts. It quantifies feature modularity and use case complexity.

The proposed modularity metrics quantify three types of relations involving features and use cases. First, *Feature Diffusion over Use Cases* (FDU) is used for quantifying how many use cases are affected by a specific feature. On the other hand, *Number of Features per Use Case* (NFU) is used for quantifying how many features are tangled within a specific use case. We assume that each use case should focus on its primary goal, although several features might be related to the primary goal of a use case. Finally, we applied the metric *Feature Diffusion over Scenarios* (FDS) in order to quantify how many internal use case members (scenarios) are necessary for the materialization of a specific feature.

Additionally, we used three metrics related to complexity. The first one, *Vocabulary Size*, quantifies the number of use cases (VSU) and scenarios (VSS) required by each of the evaluated approaches. The second one, *Steps of Specification* (SS), is related to the size of each scenario and identifies how many pairs *User action* x *System response* compose a specific scenario. Additionally, we also relate modularity to complexity by applying *Features and Steps of Specification* (FSS), which counts the number of steps of specification whose main purpose is to describe the behavior of a feature. A complete description of these metrics can be found elsewhere [6]. Table 4 summarizes the evaluation of these metrics applied to one of our case studies — a real multimedia message (MMS) product line [6].

In order to enhance our confidence, more controlled studies should be performed in this thesis. In what follows, we present possible improvements in our evaluation process.

Table 4. Modularity and complexity metrics

	PLUC	PLUSS	Crosscutting
Mean value of FDU	3.5	3.5	2
Mean value of FDS	6.25	5	4.25
Mean value of NFU	2	2	1
Mean value of FSS	12	11	10.25
VSU	5	5	7
VSS	27	24	23
SS	75	64	56

3.3 Evaluation improvements

Although we have performed some empirical studies using our approach, and compared it with existing ones, the level of control in these studies can be considered low. Therefore, performing a formal experiment is the primary improvement of our evaluation process. The goal of this experiment is twofold: confirm the benefits of a clear separation between variability models and use case scenarios; and validate (or enhance) our proposed metric suite. Due to space constraints, we just present an overview of our initial experiment design.

The experiment will be conduced as a class activity, involving undergraduate Computer Science students. They will be responsible for specifying and evolving scenarios of different SPLs. The technique used for specifying scenarios will be used as the treatment variable of our experiment [17]. Depended on the number of techniques that will be evaluated, two or three systems are going to be specified. Therefore, if just PLUSS and our approach are going to be compared, just two SPLs will be used in our experiment. On the other hand,if PLUC, PLUSS, and our approach are going to be compared, three SPLs will be used.

Students and SPLs are the two variables (or factors) that must be controlled. In order to minimize the *experimental error* motivated by such variables [17], we have to randomize the activities in such a way that each student will be assigned to each evaluated technique and SPL. An interesting layout for this initial arrangement is the *Latin Square* design [7]. The response variables will be the time needed to specify the SPLs and to perform a sequence of changes.

4 Related work

Aspect-oriented semantics. Masuhara and Kiczales propose a framework for representing the semantics of different aspect-oriented mechanisms [16]. A key characteristic of of this framework is that aspect-oriented techniques are described as a three-part description, where two input languages crosscut each other with respect to the result computation or program. In our approach, we present a slight customization of the Masuhara and Kiczales work, proposed to represent the semantics of variability management as a crosscutting mechanism. In this case, different product line models crosscut each other to derive product specific artifacts.

Variability models. Pohl et al. argue that variability management should not be integrated into existing models [18]. Their proposed Orthogonal Variability Model (OVM) describes traceability links between variation points and the conceptual models of a SPL. Our approach also decouple variability concern. However, we describe, in more details, its semantics as a crosscutting mechanism. Additionally, we make clear how to relate SPL features to software engineering assets by means of the configuration model. In our approach, such a model can also be used for reasoning about traceability.

Scenario variability. Several approaches have been proposed for representing scenario variability [15, 14, 10, 4]. However, in this work we are only comparing our approach with PLUC and PLUSS techniques, because they encompass a broad range of SoC between variability management and scenario specification. Moreover, they were propose specifically for representing variability in SPLs. It is important to notice that the primary goal of existing approaches is to provide variation points in use case scenarios. Hence, the main contribution of our crosscutting approach is that we introduce the new perspective of creating a better separation between variability management and scenario specification.

5 Concluding remarks

In this paper we presented an overview of our research, describing its objectives, achieved results, intended evaluation, and future work. In summary,

this thesis aims to evaluate the benefits of reasoning about variability management as a crosscutting concern. In order to do that, we are proposing a modeling framework that describes how *variability artifacts* (feature models, product configuration, and configuration knowledge) crosscut to *common software engineering artifacts* with respect to product line instances.

Acknowledgements

We would like to thank the anonymous referees, the Software Productivity Group, and the Motorola Brazil Test Center Research Group for making several suggestions to improve this article. This work is partially supported by the Brazilian Research Agency, CNPq, grant CT-INFO 17/2007.

References

[1] V. Alves, A. C. Neto, S. Soares, G. Santos, F. Calheiros, V. Nepomuceno, D. Pires, J. Leal, and P. Borba. From conditional compilation to aspects: A case study in software product lines migration. In *1st Workshop on Aspect-Oriented Product Line Engineering (AOPLE)*, Portland, USA, Oct 2006.

[2] S. Apel, T. Leich, and G. Saake. Aspectual mixin layers: aspects and features in concert. In *ICSE '06: Proceedings of the 28th international conference on Software engineering*, pages 122–131, New York, NY, USA, 2006. ACM.

[3] C. Baldwin and K. Clark. *Design Rules The Power of Modularity*, volume 1. The MIT Press, first edition edition, 2000.

[4] A. Bertolino and S. Gnesi. Use case-based testing of product lines. In *ESEC/FSE-11: Proceedings of the 9th European software engineering conference*, pages 355–358, New York, NY, USA, 2003. ACM.

[5] R. Bonifácio and P. Borba. Representing variability modeling as a crosscutting concern. Technical report, Informatics Center, 2008.

[6] R. Bonifácio, P. Borba, and S. Soares. On the benefits of variability management as crosscutting. In *Early Aspects Workshop at AOSD*, Brussels, Belgium, mar 2008.

[7] G. Box, J. Hunter, and W. Hunter. *Statistics for Experimenters: Design, Innovation, and Discovery*. Wiley-Interscience, 2 edition, 2004.

[8] P. Clements, L. Northrop, and L. M. Northrop. *Software Product Lines : Practices and Patterns*. Addison-Wesley Professional, August 2001.

[9] K. Czarnecki and U. Eisenecker. *Generative programming: methods, tools, and applications*. ACM Press/Addison-Wesley Publishing Co. New York, NY, USA, 2000.

[10] M. Eriksson, J. Borstler, and K. Borg. The pluss approach - domain modeling with features, use cases and use case realizations. In *International Conference on Software Product Lines*, pages 33–44. LNCS, 2005.

[11] E. Figueiredo, N. Cacho, C. SantAnna, M. Monteiro, U. Kulesza, A. Garcia, S. Soares, F. Ferrari, S. Khan, F. C. Filho, and F. Dantas. Evolving software product lines with aspects: an empirical study on design stability. In *ICSE '08: Proceedings of the 30th international conference on Software engineering*, pages 261–270, New York, NY, USA, 2008. ACM.

[12] A. Garcia, C. Sant'Anna, E. Figueiredo, U. Kulesza, C. Lucena, and A. von Staa. Modularizing design patterns with aspects: a quantitative study. *Aspect-oriented software development: Proceedings of the 4 th international conference on Aspect-oriented software development*, 14(18):3–14, 2005.

[13] J. Greenfield and K. Short. *Software factories: assembling applications with patterns, models, frameworks and tools*. ACM Press New York, NY, USA, 2003.

[14] M. L. Griss, J. Favaro, and M. d' Alessandro. Integrating feature modeling with the rseb. In *ICSR '98: Proceedings of the 5th International Conference on Software Reuse*, page 76, Washington, DC, USA, 1998. IEEE Computer Society.

[15] I. Jacobson, M. Griss, and P. Jonsson. *Software reuse: architecture, process and organization for business success*. ACM Press/Addison-Wesley Publishing Co. New York, NY, USA, 1997.

[16] H. Masuhara and G. Kiczales. Modeling crosscutting in aspect-oriented mechanisms. In *European Conference on Object-Oriented Programming (ECOOP)*, Lecture Notes in Computer Science, pages 2–28. Springer, 2003.

[17] S. Pfleeger. Design and analysis in software engineering: the language of case studies and formal experiments. *ACM SIGSOFT Software Engineering Notes*, 19(4):16–20, 1994.

[18] K. Pohl, G. Böckle, and F. J. van der Linden. *Software Product Line Engineering : Foundations, Principles and Techniques*. Springer, September 2005.