Managing Variability in Business Processes: An Aspect-Oriented Approach

Idarlan Machado
Computer Science Department
University of Brasília
Brasília, Brazil
idarlan@yahoo.com.br

Rodrigo Bonifácio
Independent Consultant
Brasília, Brazil
rbonifacio@computer.org

Vander Alves
Computer Science Department
University of Brasília
Brasília, Brazil
valves@unb.br

Lucinéia Turnes
Computer Science Department
University of Brasília
Brasília, Brazil
lucineiaturnes@gmail.com

Giselle Machado
Computer Science Department
University of Brasília
Brasília, Brazil
gisellegiba@gmail.com

ABSTRACT

Business processes specify key activities in an organization, some of which can be automated. It is often the case that replication of activities across such processes occur and failure in identifying such replication results in organizational costs. To minimize this risk and optimize organizational resources, in this paper we characterize variability in business process and propose an approach to manage such a variability. The characterization of variability relies on the study of industrial-strength applications in the Human Resources domain. The management of variability is based on a compositional and parametric approach with Aspect-Orientation. It leverages and extends an existing tool to address variability in such domain.

Categories and Subject Descriptors

D.2.1 [**Requirements-Specifications**]: Methodologies

General Terms

Management

Keywords

Software Product Lines, Business Processes, Aspects, Composition

1. INTRODUCTION

Business processes are a way for an organizational entity to organize work and resources (people, equipment, information, and so forth) to accomplish its aims [5]. These processes specify key activities, roles, and artifacts produced in a specific manner by an organization. Some activities are performed manually and others can be automated by the development of one or more systems. Such processes are essential for achieving business goals and compliance to them is an importance measure of organizational maturity [7].

Nevertheless, it has been reported that it is often the case that replication of activities or even whole processes occur and failure in identifying such replication results in unnecessary organizational costs regardless of the quality of the underlying software supporting the existing processes [9]. For example, in an organization having different branches it could happen that these branches have their own payroll systems, despite belonging to the same organization. Although each branch has its own particular processes (reflecting some possibly local legislation), it is expected that most of the processes would be similar to other branches. This replication is clearly against business goals (e.g, efficiency, maximization of resource allocation, among others).

In order to minimize this risk and optimize allocation of organizational resources, it is important first to be aware of this replication and then to handle such commonality and variability. Accordingly, in this paper we characterize variability in business process (Section 2) and then present a preliminary approach–a detailed evaluation is outside the scope of this paper–to manage such a variability (Section 3). The characterization of variability relies on the study of industrial-strength applications in the Human Resources domain. The management of variability is based on a compositional and parametric approach based on Aspect-Orientation. It leverages and extends an existing infra-structure with new transformations, modeling of relevant artifacts (business processes), their variability, and a new configuration knowledge [4] mapping features expressions to such new transformations. These transformations are responsible to generate a product in another derivation phase. Unlike our approach, the Cappelli's approach [2] does not address variability issues in business processes, not focusing on thus configuration and domain knowledge model. Related work is considered in Section 4, and Section 5 offers concluding remarks.

2. BUSINESS PROCESS VARIABILITY IN THE HUMAN RESOURCES DOMAIN

Our research is motivated by variability in the Human Resources (HR) domain, realized in the context of a research and development project. Using an extractive adoption strategy, we aim at constructing a business process product line in this domain. We briefly describe the extractive process and then characterize the variability in this model, which motives our approach for variability management presented in Section 3.

The extractive process was performed having as input existing business process models of three different organizations in the HR domain. These were specified in different notations (textual and flowcharts). We lead a group of analysts in performing the following tasks: 1) modeled such process in BPMN, resulting in fifty two processes; 2) analyzed the resulted modeled business processes for communality and variability; 3) built the domain model (feature model). The choice of BPMN was a constraint imposed by the sponsor of the project. An excerpt of the resulting feature model in the HR domain is shown in Figure 1. An instance of such a model corresponds to a collection of business processes targeted at a particular organization. For instance, in Brazil, two such configurations could be the Civil Servant (bound by Law 8.112) and CLT schemes (more common in private organizations). In the later, for instance, leave for marriage is three days, whereas for the former is eight days. Other configurations are possible for a myriad of specific public and private organizations and even countries. For instance, the *monitoring* feature is not bound by Law 8.112, but is a desirable feature to ensure the governance and quality control.

The business process commonality analysis considered the existent conceptual grouping of these process, which could be inferred by their names (recruitment, payroll, allowance, tenure-track, leave, retirement, and so on) and their corresponding activities. The variability analysis focused on identifying variability patterns involving activities within processes sharing a significant amount of similarity. A total of sixteen fine-grained variability patterns were identified, which could be further classified into the following coarse-grained patterns: 1) insert/removal/replacement of activity/flow of activities before/after/around activity/gateway/-subprocess; 2) parameter value variability within flow objects; 3) variability of lanes to which activities belong.

For instance, Figures 2 and 3 illustrate business processes corresponding to different kind of allowances in the feature model (Figure 1), *food* and *mobility*, respectively. Due to space reasons, the illustrations are simplifications of the actual processes, but are still representative of issues in the original processes. The gray areas in Figures 2 correspond to their variability: 1) the former has one gateway and an extra activity inserted after the *register* activity, when compared to the latter, in order to handle duplicate benefits; 2) the processes also differ by the value to which a parameter is bound in the *Carry out financial arrangement* activity. This parameter is defined at the *allowance* feature. Therefore, such processes exhibit variability patterns (1) and (2) mentioned above. Variations such as these were observed in among other processes as well.

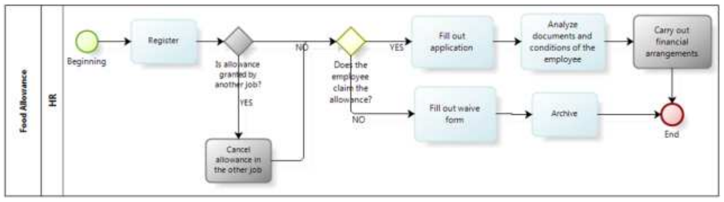

Food Allowance
HR
Beginning
Register
Is allowance granted by another job?
YES
Cancel allowance in the other job
NO
Does the employee claim the allowance?
YES
Fill out application
Analyze documents and conditions of the employee
Carry out financial arrangements
NO
Fill out waive form
Archive
End

Figure 2: Food Allowance Business Process

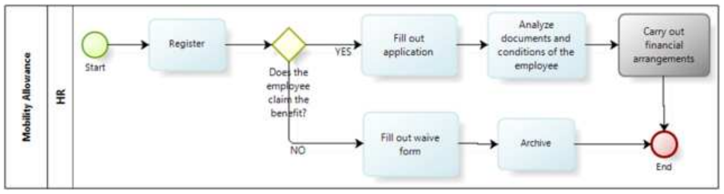

Mobility Allowance
HR
Start
Register
Does the employee claim the benefit?
YES
Fill out application
Analyze documents and conditions of the employee
Carry out financial arrangements
NO
Fill out waive form
Archive
End

Figure 3: Mobility Allowance Business Process.

3. MANAGING VARIABILITY IN BUSINESS PROCESS PRODUCT LINES

Our approach for managing product line variabilities in business processes is built upon the variability model of Modeling Scenario Variability as Crosscutting Mechanisms (MSVCM) [1], which also provides a set of Haskell libraries and tools for product line development (named Hephaestus), and thus is characterized by the following: a) considers the contribution of the configuration knowledge, a dedicated model for relating features to transformations that resolve product line variabilities in business processes; b) uses aspect-oriented constructs and parameterization, leading to a modular specification of the core and variant assets of a business domain.

We start this section by describing an BPMN extension for representing product line variability. This extension introduces a notion of *business process aspects*, which aims to modularize variability in business processes. An abstract representation of this extension is detailed in Section 3.1. After that, in Section 3.2, we detail a customization of the configuration knowledge [1], which here guides the evaluation of specific transformations that solve PL variabilities in business processes. Finally, Section 3.3 illustrates the use of our approach to manage the running example of the Human Resources Domain (Section 2).

3.1 BPMN Extensions

In order to manage variability in business processes, we first extended the core elements of BPMN, in such a way that we could represent the variant part of a process using *aspect-oriented* constructs— mainly the notion of AspectJ advice-pointcut composition— and parameterization. Second, we proposed a set of transformations for resolving variability in business processes models. These transformations comply with the signature of the transformations first introduced in [1]. For this reason, we could reuse both the structure and interpreter of the MSVCM *Configuration Knowledge*.

Here we present the BPMN extensions and transformations using the Haskell functional programming language [8], the same language used to implement *Hephaestus*. Such a decision leads to a concise definition of the product derivation phase, as we explain in Section 3.2. We extended BPMN

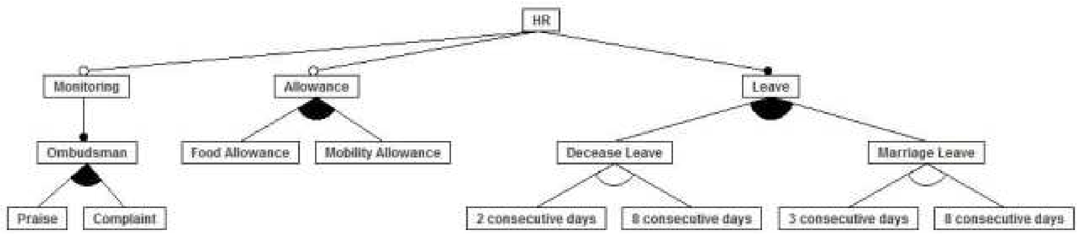


Figure 1: Feature Model of the Human Resources Domain. Unfilled circles and arches denote optional features and alternative features, respectively. Filled circles and arches denote mandatory features and or-features (at least one must be chosen), respectively.

elements by means of a small *embedded domain specific language* [6], which comprises a set of algebraic types and operators that an analyst can use to instantiate business processes using Haskell as a host language.

The notion of process type (either a basic process or advice) is our main extension to BPMN (Listing 1, lines 10–12). Usually, in our approach we model commonalities using *basic* business processes; whereas we represent flow variability using processes of the *advice* type (which is further specialized as *before*, *after* or *around*). Each advice process has a *pointcut* clause identifying the places within the business process model where the aspect composition should occur.

Listing 1: abstract syntax excerpt of BPMN extension in Haskell

```
1  data BusinessProcessModel =
2   BPM { processes :: [BusinessProcess] }
3  data BusinessProcess =
4   BusinessProcess {
5     pid :: Id,
6     ptype :: ProcessType,
7     objects :: [FlowObject],
8     transitions :: [Transition]
9  }
10 data ProcessType =
11  BasicProcess |
12  Advice {advType :: AdviceType, pc :: Pointcut}
13 data FlowObject =
14  FlowObject {
15    fId :: Id,
16    fType :: FlowObjectType,
17    annotations :: [Annotation],
18    parameters :: [Parameter]
19 } | Start | End
20 data FlowObjectType = Activity | Gateway
21 data Pointcut = PC String
22 type Transition =  (FlowObject, FlowObject, Condition)
```

We also extended BPMN so that we could assign annotations (Line 20 of Listing 1) to the *FlowObject* data type, which might be used to represent BPMN activities or gateways, in order to expose joinpoints to the set of BPMN advice. A *matches* function (Listing 2) verifies if a given flow object (f) matches the pointcut clause of an advice (adv). In this case, the *adv* objects and transitions are introduced in the business process (an operational semantics of this composition in Listing 4), either *before*, *after*, or *around* the flow object f. It is possible to extend this joinpoint model by means of introducing a new *Pointcut* constructor and providing a new implementation of the *matches* function (probably, using the Haskell pattern-matching capability).

Listing 2: A function that matches pointcuts and annotation based joinpoints in Haskell

```
1 matches :: FlowObject → Pointcut → Bool
2 matches f (PC p) = p ∈ (annotations f)
```

A final extension to BPMN is that our *FlowObject* data type is a parameterized entity (Line 21 of Listing 1). The goal here is to enable fine-grained variability, such as required by the *allowance* feature discussed in Section 2. In that case, we were not expecting to combine flow objects and transitions with an existing business process; instead, we just had to bind the values of a flow object parameter with the selected option(s) of the *allowance* feature.

3.2 Configuration Knowledge and Transformations

The *Configuration Knowledge* is a specific asset of the MSVCM approach that relates feature expressions to model transformations. In more details, a configuration knowledge corresponds to a list of *Configuration Item*, which represent a mapping of $feature\ expressions$ to $transformations$ (Listing 3). If a feature expression is satisfied by an SPL member, the related transformations are applied. In this way, the set of suitable transformations are responsible for automatically generating a product specification.

To customize the *Configuration Knowledge* (CK) to the business process domain, we just have to change the signature of the *Transformation Data Type*. As shown in Listing 3, a transformation is any function that expect two arguments (an *SPL* argument representing the SPL assets and a *Product* argument representing a product on a specific stage of the process derivation) and returns a refined version of the product — with some variability resolved.

Therefore, our CK representation maps feature expressions to transformations that might select or configure business processes for a specific product configuration. Distinct transformations deal with the types of variability discussed in Section 2 and indeed take additional parameters. Nevertheless, since these functions are curried, their partial application leads to functions that obey the *Transformation* signature. The mentioned transformations are as follows:

(a) **Select Business Process:** As we show in the code snippet bellow, the *selectBusinessProcess transformation* first create a list *bps* which comprises the *spl processes* whose identifiers equal *bpId* (the first argument) and that are not present in the *product processes*. After that, this transformation concatenates the the *bps processes* with the processes already selected in the product.

```
selectBusinessProcess bpId spl product =
 let bps = [bp
           | bp ∈ (processes spl)
           , pid bp == bpId
           , bp ∉ (processes product)]
 in  product {productBpm = [bps] ∪ (processes product)}
```

(b) **Evaluate Advice:** the *evaluateAdvice* transformation first obtains a list of advice (*advs*) whose identifiers equal *advId*. Our type checker (not covered in this paper) do not allow more than one *advice* or *basic* business process sharing the same advice. For this reason, we just match *advs* with a list of a single element (*[adv]*) or the empty list (*[]*). In the first case, the resulting business process model of the product is computed by the application of the *eval* function to all process of the product. In the second case (empty list), we just return the product without applying any transformation. The *eval* function just call the proper evaluation of a before, after, or around advice.

```
evaluateAdvice advId spl product =
 let advs = [a | a ∈ (processes spl)
            , (pid a) == advId]
 in case advs of
  [adv] → product {productBpm = wovenProcesses }
  [] → product
 where wovenProcesses =
  map (eval adv) (processes product)

eval adv bp =
 in case ptype adv of
   BasicProcess      → bp
   (Advice After  _) → evaluateAfterAdvice adv bp
   (Advice Before _) → evaluateBeforeAdvice adv bp
   (Advice Around _) → evaluateAroundAdvice adv bp
```

(c) **Bind Parameter:** The next code snippet details the *bindParamter* transformation. It first obtains the selected options of the feature identified by the *fId* argument. For a given feature *f*, these options correspond to the selected sub-features (children) of *f* in a product configuration. Second, the *bindParameter* transformation computes a *string* representation of the selected options; and finally it bind this value to the flow objects parameters identified by *pId*.

```
bindParameter pId fId spl product =
 let
  options = concat [children f | f ∈ (pc product)]
  optionsStr = concat [name o | o ∈ options]
 in
  product {process = boundProcesses}
 where
  boundProcesses =○ ..
```

Listing 3: Configuration Knowledge Extension in Haskell.

```
type ConfigurationKnowledge = [ConfigurationItem]
data ConfigurationItem = ConfigurationItem {
  expression :: FeatureExpression,
  transformations :: [Transformation]
}
type Transformation = SPL → Product → Product
data SPL = SPL {
 fm :: FeatureModel,
 splBpm :: BusnessProcessModel
}
data Product = Product {
 pc :: ProductConfiguration,
 productBpm :: BusinessProcessModel
}
```

3.3 Revisiting Variability in the Human Resources Domain

Considering the artifacts of the Human Resources Domain (Section 2), this section details how we could manage business process variability using the transformations just presented.

First of all, we have to separate the common and variant behavior of the business processes shown in Figure 3 and Figure 2. As dicussed in the previous sections, we create (a) a *basic* business process with the shared flow objects and transitions; and (b) an *advice* formed by the the variant assets— in this case, an adtitional gatway and a new activity. Figure 4 presents the resulting business processes.

Next, to enable the configuration of the common business process, we have to relate the feature expression *Allowance* to the transformation *selectBusinessProcess "bpCommonAllowance"*, where *bpCommonAllowance* is the identifier of the business processes that handles the commonality among the allowance processes. In a similar way, in order to evaluate the *Food Allowance* advice (Figure 4), we have to relate the feature expression *Food Allowance* to the transformation *evaluateAdvice "advFoodAllowance"*, where *advFoodAllowance* is the identifier of the *Food Allowance* advice declared in the SPL assets. Figure 4-(c) shows a fragment of the configuration knowledge with these transformations.

Therefore, the product derivation process applies the transformations that are related to the valid expressions for a product configuration. For instance, if a product is configured with the *Allowance* and *Food Allowance* features, the product will comprise a business process formed by the composition of the *Allowance* business process and the *Food Allowance* advice, as Figure 2 shows. Differently, if the product is not configured with the *Food Allowance* feature, the mentioned composition will not happen, since the *evaluateAdvice "advFoodAllowance"* transformation will not be evaluated.

To sum up, the product derivation process evaluates whether a feature expression is valid or not for a specific product, generating a list with the related transformations that should be applied. After that, it calls each transformation iteratively, resolving the variabilities of the input product.

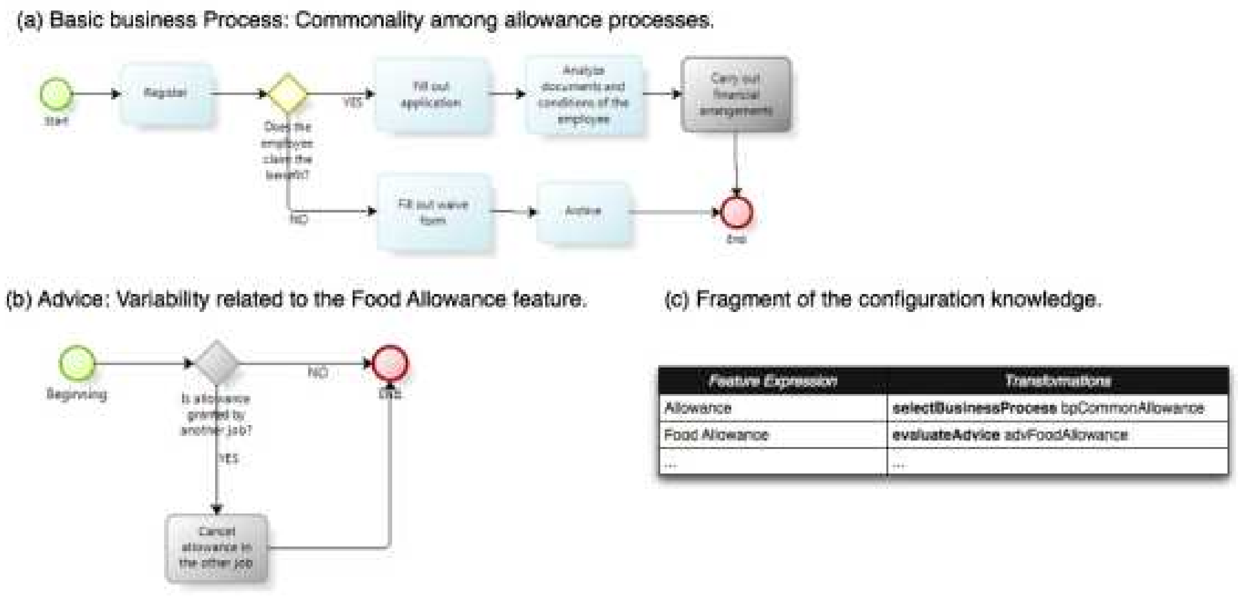
4. RELATED WORK

Variability management [11, 10] in software product lines [3] has been described at various level of abstractions. In particular, La Rosa et al. [9] propose a method and tool suite for developing processes based on configurable process model. This proposal addresses variability in connectors, gateways, activities, organizational resources. Unlike our approach, which is compositional, their approach is annotative, basically wrapping the variability with additional gateways driven by the decision model, cluttering the process model with configuration knowledge details, and hence not providing means for separating the solution and configuration spaces [4] and hindering reusability of the variability.

Cappelli et al [2] propose an aspect-oriented approach to modularize crosscutting concerns in business process model-

Listing 4: An interpreter-based, operational semantics of the evaluate after advice composition in Haskell.

```
evaluateAfterAdvice :: BusinessProcess → BusinessProcess → BusinessProcess
evaluateAfterAdvice adv bp = bp {
     objects = nub ((objects bp) ∪ (objects adv)),
     transitions = (
       [(i,j,k) | (i,j,k) ∈ (transitions bp), not (i ‘matches‘ (pointcut adv))] ∪
       [(i,y,z) | (i,j,k) ∈ (transitions bp), (x,y,z) ∈ startTransitions adv, i ‘matches‘ (pointcut adv)] ∪
       [(x,j,z) | (i,j,k) ∈ (transitions bp), (x,y,z) ∈ endTransitions adv, i ‘matches‘ (pointcut adv)] ∪
       [(x,y,z) | (x,y,z) ∈ (transitions adv), (x, y, z) ∉ ((startTransitions adv) ∪ (endTransitions adv))])
}
-- auxiliary functions
startTransitions :: BusinessProcess → [Transition]
startTransitions bp = [(i, j, k) | (i, j, k) ∈ (transitions bp), i == Start]
endTransitions :: BusinessProcess → [Transition]
endTransitions bp = [(i, j, k) | (i, j, k) ∈ (transitions bp), j == End]
```


(a) Basic business Process: Commonality among allowance processes.
Start
Register
Does the employee claim the benefit?
YES
Fill out application
Analyze documents and conditions of the employee
Carry out financial arrangements
NO
Fill out waive form
Archive
End
(b) Advice: Variability related to the Food Allowance feature.
Beginning
Is allowance granted by another job?
NO
End
YES
Cancel allowance in the other job
(c) Fragment of the configuration knowledge.

Feature Expression	Transformations
Allowance	**selectBusinessProcess** bpCommonAllowance
Food Allowance	**evaluateAdvice** advFoodAllowance
...	...

Figure 4: Separating common and variant processes using our approach.

ing. They propose an extension of BPMN to express crosscutting concerns through the insertion of crosscutting processes (represented by activities) and crosscutting relationships (represented by flows among activities). Unlike our approach, their approach does not address variability issues in business processes, thus not focusing on configuration knowledge and domain model. We further provide operational semantics of the advice types by leveraging and extending an existing functional infra-structure, including a tool.

5. CONCLUSION

We have characterized variability in business process and propose an approach to manage such a variability. The management of variability is based on a compositional and parametric approach with Aspect-Orientation. It leverages and extends an existing tool to address variability in such domain. The approach has not been applied to production nor evaluated within the industrial partner yet, but the organization has reacted positively to the proposed language as a means to decrease replication. As future work, we will finish the implementation for handling variability in lanes in business processes. We are also finishing the implementation of the around advice and of support for quantification. Further, we plan to conduct a thorough empirical evaluation in the HR domain.

6. ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for valuable suggestions to improve this work.

7. REFERENCES

[1] R. Bonifácio and P. Borba. Modeling scenario variability as crosscutting mechanisms. In *Proc. of AOSD '09*, pages 125–136. ACM, 2009.
[2] C. Cappelli, J. Leite, T. Batista, and L. Silva. An aspect-oriented approach to business process modeling. In *Proc. Early Aspects*, pages 7–12. ACM, 2009.
[3] P. Clements and L. M. Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2002.

[4] K. Czarnecki and U. Eisenecker. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley Professional, June 2000.

[5] M. Dumas, W. van der Aalst, and A. H. ter Hofstede. *Process-aware information systems: bridging people and software through process technology*. John Wiley and Sons, 2005.

[6] P. Hudak. Building domain-specific embedded languages. *ACM Comput. Surv.*, 28, December 1996.

[7] J. Jeston and J. Nelis. *Business process management: practical guidelines to successful implementations*. Butterworth-Heinemann, 2006.

[8] S. P. Jones et al. The haskell 98 report (revised). Technical report, Cambridge University Press, 2002.

[9] M. L. Rosa, M. Dumas, A. ter Hofstede, and J. Mendling. Configurable multi-perspective business process models. *Information Systems*, 26(2), 2011.

[10] M. Svahnberg, J. van Gurp, and J. Bosch. A taxonomy of variability realization techniques. *Softw., Pract. Exper.*, 35(8):705–754, 2005.

[11] J. van Gurp, J. Bosch, and M. Svahnberg. On the notion of variability in software product lines. In *WICSA*, pages 45–54, 2001.