Electronic Communications of the EASST
Volume 8 (2008)

Proceedings of the Third International ERCIM Symposium on Software Evolution (Software Evolution 2007)

A Requirements-Based Taxonomy of Software Product Line Evolution

Klaus Schmid, Holger Eichelberger

13 Pages

Guest Editors: Tom Mens, Ellen Van Paesschen, Kim Mens, Maja D'Hondt
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

A Requirements-Based Taxonomy of Software Product Line Evolution

Klaus Schmid, Holger Eichelberger

Software Systems Engineering. Universität Hildesheim
Marienburger Platz 22, D-31141 Hildesheim
{schmid, eichelberger}@sse.uni-hildesheim.de

Abstract: Software product lines are, by their very nature, complex software systems. Due to the interconnectedness of the various products in the product line any form of evolution becomes significantly more complex than in a single system situation. So far most work on product line evolution has focused on specific approaches to supporting special cases of the evolution problem. In this paper, we take a different approach and provide a broad taxonomy of requirements-driven evolution in software product lines. This serves as a basis for the identification of requirements on evolution support.

Keywords: Software Product Lines, Evolution, Traceability, Requirements

1 Introduction

Software product line (SPL) engineering [5, 11] is an important approach to the efficient development of large numbers of software systems that promises major improvements in terms of time, costs, and quality based on large-scale reuse. Experience shows that product development costs can be reduced by 80% and that cost of quality can be reduced to 50% [11]. However, as a result, the various products in a SPL become interconnected: they share various assets and any change that is relevant to one product may actually have ramifications for several other products due to product interdependencies. This range of shared assets is also called the product line infrastructure and covers the whole range of product development starting from requirements through implementation and up to test assets.

Product Line Engineering

Software product line development consists of two main development cycles: domain engineering, which aims at developing software assets *for reuse* and effectively establishes the product line infrastructure, and application engineering. Application engineering aims at development *with reuse* and develops the final products based on the product line infrastructure. This is shown in Figure 1. The distinction between application engineering and domain engineering is also referred to as the *two-lifecycle model*.

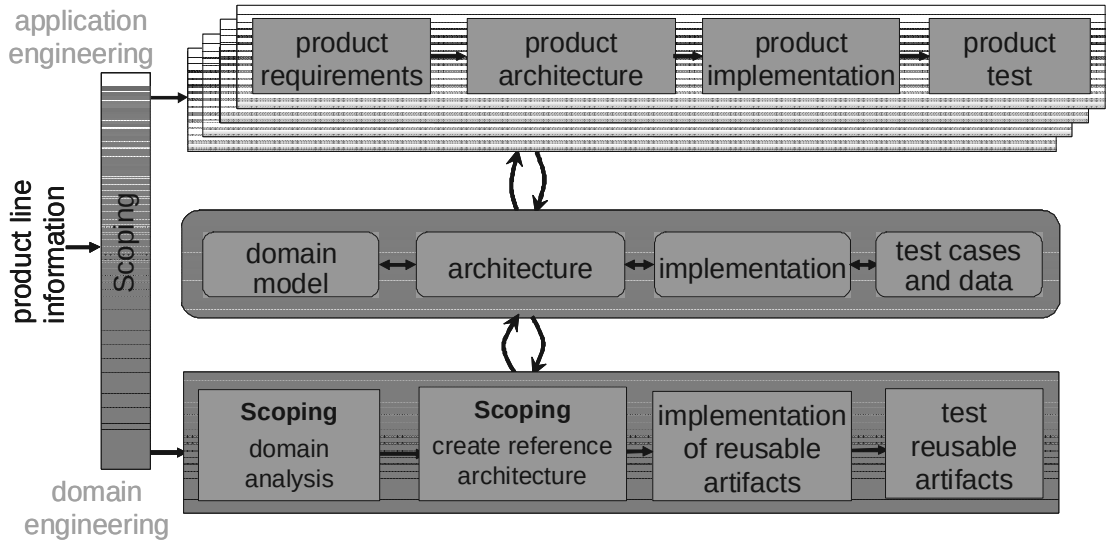

Figure 1 Structure of Product Line Engineering

The assets in the product line infrastructure may amount to over 80% of all assets in the final product [11]. As the assets are reused over and over, a strong relationship among the various products is established, because the products share these assets.
This interconnection of the various product developments through the product line infrastructure leads to a strong need for adequate support for SPL evolution. The evolution problem is further increased as a SPL needs to integrate all the changes relevant to any product it supports and the SPL as a whole is usually much longer-lived than an individual product [18]. The combination of these issues leads to a situation where the evolution problem is much more severe than in single system development.

Requirements-Driven Evolution

In this paper, we focus on requirements-driven SPL evolution, i.e., those evolution tasks, that can arise in a SPL context based on new or changed requirements. Typical examples are the introduction of new products that must be supported or the modification of requirements of existing products. The reason for this focus is that we are particularly interested in supporting requirements engineering for software product lines. Classical treatment of software evolution prefers the categorization of perfective, corrective, and adaptive maintenance [8] (sometimes also preventive). Usually, requirements changes belong to the categories of perfective or adaptive maintenance.
Due to our requirements focus, we will in particular refrain from addressing issues relating to later stages of the development process, like refactoring the product line infrastructure. This results in a taxonomy which is necessarily narrower than general classifications of software change [4]. We regard this taxonomy as a means for categorizing requirements changes in a product line context. In a second step, we aim at identifying the support that is required by requirements engineering tools in order to provide adequate requirements engineering for software product lines.

In section 2, we will introduce a basic terminology for SPL and provide a high level discussion of different possible situations for product line evolution as a basis for our terminology. This will provide the basis for our refined discussion in section 3, where we will summarize our requirements for evolution support for SPL. In section **Error! Reference source not found.**, we will discuss related work from the area of evolution in general and from the area of SPL evolution in particular. Finally, in section 5 we will conclude.

2 Evolution Categories and the Evolution Taxonomy

In this section, we introduce the basic categories of our taxonomy. In the following subsections we will describe the taxonomy itself by discussing each of these basic categories in detail.
Roughly, requirements-driven evolution can result in changes on three levels:

- **Requirements level change** – change to an individual requirement (or small group of requirements).
- **Product level change** – change in terms of products that must be supported.
- **Product line level change** – change to whole groups of products that must be supported.

As a basis for our discussion we will categorize requirements of a product in a SPL in three categories:

- **Commonalities** – requirements that are common to all products in the SPL.
- **Variabilities** – requirements that are not common for all products, but there are systematic variations among the products, i.e., a variability is usually relevant to multiple products.
- **Product-Specific** – these are requirements that are relevant only to an individual product.

In general, the various products will include requirements from all three categories, i.e., the resulting set of requirements will usually be composed of commonalities, selected variabilities and product-specific requirements. The product line infrastructure (i.e., the reusable artifacts) will contain only commonalities (including the implementation) and some variabilities. The product-specific parts (requirements and implementation) are only regarded as part of the products.
We would like to illustrate this with examples from a hypothetical online shop product line. We will assume that all instances of the product line, i.e., all products, are based on the shopping cart metaphor. Thus, the shopping cart is a commonality of the product line. The product line will support a number of different database systems. More precisely, each product in the product line will support exactly one out of a number of database systems. The database system is thus a variability (more precisely an alternative). Further, there might be a specific product X that will support a proprietary ERP-system. The requirements for this ERP-system would then be product-specific to product X.
In the context of a hierarchical product line, the variabilities of one product line can be the commonalities of another one. We do not make this case explicit here, as we will always use the perspective of a single product line. In this case, we assume the categorization is complete.
As the basis for our analysis, we must make certain assumptions, regarding the way variability in software product lines is modeled. Here, we base our terminology on a decision-based framework similar to the one described in [17], however, our results can be easily transferred to feature-based modeling approaches like [6]. In particular, we will distinguish between a *variability* (e.g., an alternative) and a *variability resolution* (e.g., a specific case in an

alternative). Further, we will denote the usage of a specific alternative in a context as a *variability usage*. Individual variabilities may be related by means of *constraints*. Going back to the web shop example, the possible databases Oracle, MySQL, etc. would be variability resolutions. If a specific product uses a MySQL database, this would constitute a variability usage. A constraint could be that a certain database system might not be compatible with choosing a low-cost solution.
As described above, we should differentiate among changes on the levels of individual requirements, changes on the level of products and on the level of product lines as a whole. Roughly, the different changes can be subdivided into addition, modification, and deletion. In the remainder of this section, we will describe our taxonomy according to the three change levels and within each level along the subdivisions introduced above in this section. For example, if we change the way shopping carts are defined, we would have a product line modification, if we add a new product for customer Y, we have a product addition. Finally, if we delete requirements of the proprietary interface mentioned above, this is a requirements deletion.

3.1 Requirements Level Changes

These changes can be categorized as adding, deleting or modifying individual requirements (or groups of requirements). Below, we further discuss their implications

- *Adding requirements* – adding requirements can happen for all three types of requirements: product-specific, variability addition, or commonality. Product specific additions have no ramifications on other products or the product line infrastructure (except for the fact that the addition must be conformant to restrictions of the product line infrastructure). The addition of a variability or of a variability resolution (to an existing variability) may have an impact on other products as well, but only if they are selected. Adding a commonality, however, will always impact the product line infrastructure and thus have ramifications on other products as well.
- *Deleting requirements* – again this can happen on any of the levels: product-specific, variability, or commonality. For product-specific requirements there are no ramifications for other products. Variability and commonality deletions have effects on the product line infrastructure. Deletions of commonalities or variabilities that are relevant to other products also have an impact on these other products. These effects of changes need to be identified and analyzed as well.
- *Modifying requirements* – modifying requirements can happen on any of the levels: product-specific, variability, or commonality. In particular, a modification may change the category of the requirement as shown in [1]. Figure 2 depicts this fact.
    - *Product-specific modifications* can lead to a mere modification of content or can also change a requirement into a variability.
    - *Variability modifications* can change a variability into a commonality or a product-specific requirement. In particular, it needs to be determined what happens to other products that share the variability: the change may affect either all products in the same manner or it may affect only some products, for example, leading to the introduction of an additional variability resolution. Particular forms of variability modifications include the modification of the

relevant constraints or if attributes of the variability like its binding time are modified. Based on the web shop example, a typical case might be customer Y requires the use of database system A which is so far not supported by the product line. We may either add this as a variability resolution, modifying the variability on a product line wide level or we may even treat the variability database support henceforth as product-specific.

- *Commonality modifications.* Again this may either impact the commonality per se, changing it for all products. Or it may affect only some, and thus may turn the commonality into a variability. An example would be to support one-click shopping besides the cart metaphor. This would turn the corresponding commonality into a variability.

An overview of these types of changes is shown in Figure 2. Some changes may also modify the type of requirement in terms of commonality, variability, and product-specific. In Figure 2 the boxes represent the SPL requirement categories as introduced in section 2. The arrows visualize the intent of the change to a requirement, e.g. to make a product-specific requirement more generic and thereby to turn it into a variability. Furthermore, the arrows denote the impact in terms of the number of products affected by changing a requirement.

3.2 Product-Level Change

Product-level changes describe additions and deletions of whole products. Modifications of individual products can be described as requirements-level change. As defined in [9], there is a need to distinguish between the marketed and the engineered product line in product line engineering. A marketed product line denotes a set of products that are marketed together as sharing a common set of features, whereby different products often substitute each other (e.g. different versions of the Windows operating system). An engineered product line is designed and developed for and with reuse in order to share major parts of the implementation. Conversely, different products in an engineered product line may belong to different marketed product lines (e.g. navigation systems of a single producer may be sold under multiple brands). Here, we focus on the engineered product line only.

- *Adding a product* – this will require the definition of the selected variabilities and of product-specific concerns. Thus, this only establishes uses relations for the various variabilities (and commonalities).
- *Deleting a product* – this will lead to the deletion of the corresponding product-specific requirements. Usually the variabilities remain in existence in the product line infrastructure, even if they are no longer used by any product.

Modifying a product directly maps to modifying requirements as discussed above.

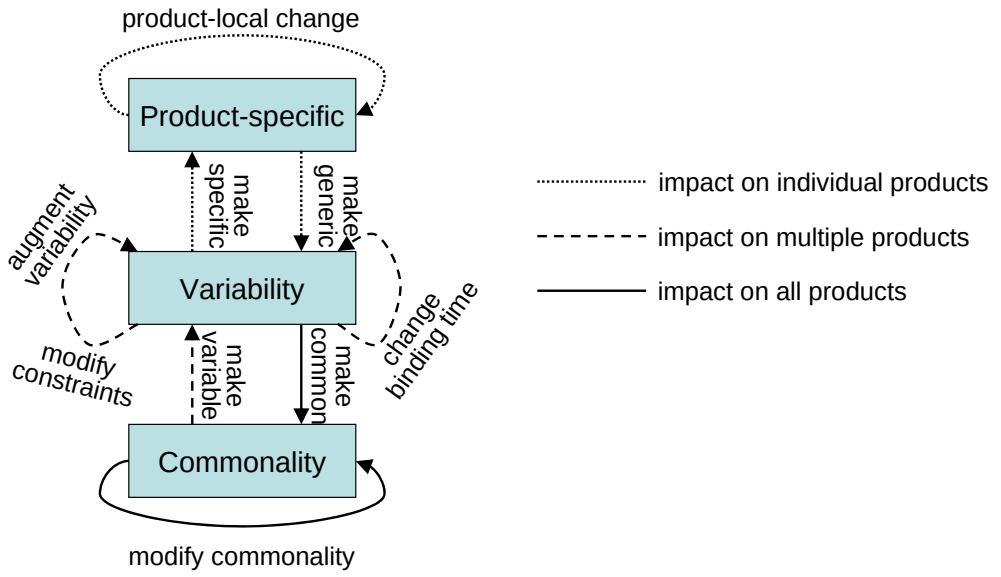

Figure 3: Summary of Requirements-Level Changes.

3.3 Product-Line-Level Change

A complex product portfolio is sometimes represented in terms of a number of product lines. These product lines can be either independent in the sense that they do not explicitly share any product line level assets or they can be structured in the sense that they share some assets. The later is sometimes also called a *product population* or a *hierarchical product line* [21].
As a consequence, we can distinguish the following cases as a refinement of the product / product line operations that were defined in the context of [1]:

- *Adding a product line* – a new product line is established. This may either be a completely new and independent product line or it is a sub product line in the sense of a product population as defined above. In the later case, there is a relation between the sub product line and the upper level product line, which is similar to the relation between an individual product and its product line.
- *Removing a product line* – the product line is removed. Again this can be subdivided into the cases that either an independent product line or a product line as a sub product line of a product population is removed.
- *Merging two product lines* – sometimes it is important to merge different product lines (e.g., in the case of an acquisition, or if they become increasingly similar over time).
- *Splitting a product line* – the opposite operation is the split of a product line. This may either lead to two independent product lines or it leads to two product lines, which are part of a product population. In fact, this case can be handled by moving the appropriate product-specific elements into the target SPL while variabilities and commonalities are simply duplicated. A variation of this is to split a single product line into a hierarchical product line. In this case, a product line infrastructure is treated as a sub-product line of another product line.

It is very difficult to formalize completeness of a taxonomy like the one given above. Thus, we can argue here only that we capture a complete set of levels (requirement, product, product line) and a complete set of actions (add, modify, delete). From this perspective, we can assume that our taxonomy is complete in the sense that any real requirements-based product line evolution can be described as a combination of instances of our taxonomy.
A different issue is, however, whether the sub-cases that we identified above are complete. This is much harder. So, at this point, we can only argue that this is a reasonably balanced categorization for the real-world situations, we have experienced so far.

3 Requirements for Evolution Support of Software Product Lines

In the preceding sections we discussed thoroughly our requirements-based SPL evolution taxonomy and categorized the different possible situations that may occur in product line evolution. Now, we will discuss requirements on evolution support for product line requirements engineering that can be derived from this taxonomy. The underlying assumption of our approach is that using a systematic characterization of changes provides a higher potential of achieving a complete and consistent set of evolution support requirements than an ad-hoc approach does.
In the following table, we summarize major operators that must be supported for product line evolution. These operators correspond to the various cases identified in the previous section. In addition, the table describes traceability relations that are relevant to supporting the described operations and go beyond the usual relationships in single system development.

Situation	Evolution Operator	Traceability Information
Requirements Level		
Add Requirement	Add product-specific requirement	---
	Add new variability	relation between product requirements and the infrastructure relation between this variability and other ones
	Add new resolution in existing variability	constraints among variability resolutions; relation between product requirements and the infrastructure
	Add new commonality	relation between product requirements and the infrastructure relation between new commonality and other requirements in the product line infrastructure
Modify Requirement	Modify content of product-specific requirement	relation among product-specific requirements relation between modified requirement and product line infrastructure.
	Modify content of variability for some products (either	relation among variability resolutions; relation between product-specific requirements

	introduces additional resolutions or introduces product-specifics)	and infrastructure requirements
	Modify content of variability for all products	relation between product and infrastructure; relation to usage of this variability resolution
	Modify variability constraints	relation to usage of variability
	Modify variability properties (e.g., binding time)	relation to usage of variability
Delete Requirement	Delete product-specific requirement	relation to other product specific and product line infrastructure requirements
	Delete variability	relation to usage of variability
	Delete variability resolution	relation to usage of variability resolution
Product Level		
Add Product	Add a product	relation to commonalities and selected variability resolutions relation between product and product-specific requirements
Delete Product	Delete a product	relation to commonalities and selected variability resolutions relation between product and product-specific requirements
Product line Level		
Adding a product line	Add an independent product line	---
	Add a sub product line	relation to commonalities and selected variability resolutions (of higher level SPL) relation of product line to specific commonalities and variabilities
Removing a product line	Remove a product line	relation to used requirements from the product line infrastructure
	Remove a sub product line	relation to commonalities and selected variability resolutions (of higher level SPL) relation of product line to specific commonalities and variabilities
Merging two product lines	Merge two independent product lines	---[1]

[1] While traceability information among the different product lines would be very useful for merging, we cannot assume that it exists, as by definition these are independent product lines.

	Merging two sub product lines	relation to commonalities and selected variability resolutions (of higher level SPL)
Splitting a product line	Split product line	relation to commonalities and selected variability resolutions (of higher level SPL)

While we cannot show in a formal manner that our taxonomy and requirements derivation is complete, we believe that the systematic analysis we performed provides a sound basis for supporting requirements-based product line evolution. We can also expect the results to be more complete then the results of other, less systematic, approaches. The table provides an overview of the operations that should be supported, as well as the traceability information that is required by the engineer in order to perform the operations.

While the systematic analysis given above provides a complete basis, it is, due to the formalistic approach, not well tuned to practical support. For example, in practice certain combinations of operations might be common, thus specific support of these operations might be useful. Some of these practical considerations are:

- It would be useful to be able to work with multiple versions of a variability model in order to enable products to rely on different versions and defer update of products untill an appropriate time.
- The difference between requirement and realization (e.g., required binding time vs. established binding time) shall be represented.

Despite these practical shortcomings, we assume that our list of requirements provide a solid and comprehensive basis for evolution support. In particular, this analysis goes for the specific issue of product line evolution beyond other existing analyses.

4 Related Work

In this section, we will first discuss software evolution in general, and then we will focus on SPL evolution in particular. In single system development evolution scenarios are typically categorized as perfective, corrective, and adaptive maintenance [8]. A general taxonomy of software change is given in [4]. There is also considerable amount of work on the classification of evolution of individual artifacts, e.g. for:

- requirements (clarification, retraction, splitting and merging) [15].
- scenarios in the sense of use cases [2]. Several static relationships like *contained in*, or *precondition of* between scenarios as well as dynamic operations on scenarios were identified. Some inter-scenario operations are *split into others*, *specialization*, *extension*, *consolidation* and some intra-scenario operations are *inclusion*, *modification* and *deletion*.
- goals in requirements engineering [16]
- design diagrams like the taxonomy of changes for UML diagrams in [3].
- source code, like [7] or [12] from the viewpoint of traceability links.

For us the work on requirements, scenarios and goals is most important as it shares the requirements-centric viewpoint we use here.

Besides this general work on software evolution, significant work has also been done in the area of software product line evolution. In the next paragraph, we briefly describe general classifications of product line evolution; in the remainder of this section we discuss work on taxonomies for SPL evolution that is closely related to our work.
In [13], the evolution of domain concepts relevant to a product line was categorized as either vertical, domain-specific evolution (i.e. extension by new features) or horizontal evolution of domain assets (e.g. for improving application functionality). In [14] the evolution of features, variation points and dependent artifacts are described using basic change operations like addition, deletion and modification, thus using the same base operations that we apply.
In [19] Svahnberg and Bosch a taxonomy for the evolution of SPL was provided that centers on the different kinds of artifacts. This taxonomy is based on two case studies:

- requirements evolution, i.e. decision for a new SPL, introduction of a new product, improvement of functionality, extension to the support of standards, changes in the version of the infrastructure or improvements to quality attributes.
- architectural evolution (e.g. split of a SPL, derivation of sub-SPLs by branching or changes to entire architecture components).
- component evolution in terms of the assumed underlying architectural framework, e.g. by adding or changing implementations of the framework interfaces.

Several differences exist between the taxonomy provided in [19] and our work. Their view is broader as it goes beyond requirements. On the other hand they provide only some relevant aspects while we introduce a taxonomy based on change levels and therefore on the impacts of the changes.
Probably the approach closest to ours was presented in the context of the ConIPF methodology [10]. Their change operations were classified as follows:

- basic change operations that cannot be further partitioned and therefore represent a single modification like addition, deletion, and modification of artifacts. Each category was then further separated according to individual artifacts.
- complex change operations which are composted of several basic operations or include additional knowledge about the modification.

Obviously, the ConIPF approach shares the basic operations with our approach (and the one in [14]), but in the focus was on artifacts while we consider requirement change levels for the categorization.
While the various descriptions of product line evolution provided significant contributions, none so far provided a consistent categorization of product line change on all three levels: ranging from individual requirements over products to product lines.

5 Discussion

We provided a systematic discussion of different product line evolution operations from the viewpoint of requirements engineering. A taxonomy of evolution operations was given, which is structured into requirements-level, product-level and product-line level changes. Based on the discussion in this paper, we outlined the traceability information influenced by the

individual evolution operations and derived a set of requirements to be fulfilled by a product line environment in order to support the systematic evolution of SPL.
To our knowledge, so far no support has been provided which is in accordance with all these requirements. We thus stipulate that the presented taxonomy can be used as a reference for product line evolution support that goes beyond existing levels of support.
Some interesting issues for further research are: to realize a prototypical product line environment for the systematic support of evolution that supports the operations and requirements outlined above, to use the prototypical environment to practically validate the taxonomy and its completeness and to evaluate SPL case studies e.g. to derive the frequency of occurrence of the individual evolution operations of our taxonomy.

References

[1] G. Böckle, P. Clements, J. McGregor, D. Muthig, and K. Schmid. A cost model for software product lines. In *Proceedings of the 5th International Workshop on Product Family Engineering (PFE'5)*, number 3014 in LNCS, pages 310–316. Springer, 2003.

[2] K. K. Breitman, J. Cesar Sampaio do Prado Leite, and D. M. Berry. Supporting scenario evolution. *Requirements Engineering*, 10(2):112–131, 2005.

[3] L. C. Briand, Y. Labiche, and L. O'Sullivan. Impact analysis and change management of UML models. In *Proceedings of Software Maintenance 2003, IEEE Conference on Software Maintenance*, pages 256–265. IEEE, IEEE Computer Society, September 2003.

[4] Jim Buckley, Tom Mens, Matthias Zenger, Awais Rashid, and Günter Kniesel. Towards a taxonomy of software change. *Journal of Software Maintenance and Evolution: Research and Practice*, 17(5):309–332, 2005.

[5] P. Clements and L. Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley, Boston, MA, 2002.

[6] K. Czarnecki, H. Chang, P. Kim, and K. Kalleberg. Feature models are views on ontologies. In *Software Product Line Conference, 2006 10th International*, pages 41–51, 21-24 Aug. 2006.

[7] J. Gustavsson. A Classification of Unanticipated Runtime Software Changes in Java. In *Proceedings of Software Maintenance 2003, IEEE Conference on Software Maintenance*, pages 4–12. IEEE, IEEE Computer Society, 2003.

[8] L. Hatton. How accurately do engineers predict software maintenance tasks? *IEEE Computer*, 40(2):64–69, February 2007.

[9] A. Helferich, K. Schmid, and G. Herzwurm. Reconciling marketed and engineered software product lines. In *Proceedings of the 10th International Software Product Line Conference*, pages 23–27, 2006.

[10] L. Hotz, K. Wolter, T. Krebs, S. Deelstra, M. Sinnema, J. Nijhuis, and J. MacGregor. *Configuration in Industrial Product Families - The ConIPF Methodology*. Akademische Verlagsgesellschaft Aka, Berlin, 2006.

[11] F. van der Linden, K. Schmid, and E. Rommes. *Software Product Lines in Action - The Best Industrial Practice in Product Line Engineering*. Springer, 2007. http://www.spl-book.net/.

[12] J. I. Maletic, M. L. Collard, and B. Simoes. An XMLBased Approach to Support the Evolution of Model-to-Model Traceability Links. In *Proceedings of TEFSE 2005, November 8, 2005, Long Beach, California, USA*, 2005.

[13] J. Meinecke, M. Gaedke, and M. Nussbaumer. A web engineering approach to model the architecture of inter-organizational applications. In Turowski and Zaha [20], pages 125–137.

[14] T. Myllymäki. Variability management in software product lines. Technical report, Tampere University of Technology, Software Systems Laboratory, 2001.

[15] C. Potts and K. Takahashi. An active hypertext model for system requirements. In J. C. Wileden, editor, *Proceedings of the 7th International Workshop on Software Specification and Design*, pages 62—68. IEEE Computer Society, December 1993.

[16] C. Rolland, C. Salinesi, and A. Etien. Eliciting gaps in requirements change. *Requirements Engineering Journal*, 9:1–15, 2004.

[17] K. Schmid and I. John. A Customizable Approach To Full-Life Cycle Variability Management. *Science of Computer Programming*, 53(3):259–284, 2004.

[18] K. Schmid and M. Verlage. The economic impact of product line adoption and evolution. *IEEE Software*, 19(6):50–57, 2002.

[19] M. Svahnberg and J. Bosch. Evolution in software product lines. *Journal of Software Maintenance*, 11(6):391–422, 1999.

[20] K. Turowski and J. M. Zaha, editors. *Component-Oriented Enterprise Applications*, Lecture Notes in Informatics, 2005.

[21] R. van Ommering. Software reuse in product populations. *Software Engineering, IEEE Transactions on*, 31(7):537–550, July 2005.