MOBILE COMMUNICATORS FOR DISABLED PEOPLE

Miguel A. Laguna and Bruno González-Baixauli
University of Valladolid, Spain

Keywords: Software Product Line, Feature Model, Mobile System, Communicator.

Abstract: Software product lines are a proven development paradigm in industrial environments. However, its application in small organizations is not easy. Our approach uses the package merge mechanism of the UML 2 meta-model as representation of the variability in the product line. The structure of the variability models is directly reflected in the relationships between packages in the architectural models, so that the traceability of configuration decisions is straightforward. A similar strategy is applied at the implementation level, using packages of partial classes. The combination of these techniques and the conventional IDE tools make the developments of product lines in small organizations easier as it removes the need for specialized tools and personnel. This article reports a successful experience with a communicator product line case study, representative of the mobile systems domain. People with certain communication problems can use these systems as a low-cost help in their everyday life. As problems vary from a person to another, a communicator product line is the indicated solution, allowing the adequate personalization of the final application to the disability of each concrete person.

1 INTRODUCTION

Software product lines (SPL) are a proven reuse approach in industrial environments, due to the combination of a systematic development and the reuse of coarse-grained components that include the common and variable parts of the product line (Bosch, 2000). However, this approach is complex and requires a great effort by the companies that take it on. The research we carry out in the GIRO research group aims to simplify the change from a conventional development process into one that benefits from the product line advantages in small and medium enterprises (SME) or organizations. For this reason, we have proposed, among other initiatives, an adaptation of the Unified Process to include specific techniques of Product Line Engineering in a process parallel to Application Engineering (Laguna et al., 2003).

As specific SPL development techniques, we must pay special attention to the variability and traceability aspects at each abstraction level. We need models that represent the product line and a mechanism to obtain the configuration of features that represent the best combination of variants for a specific application. Additionally, we must connect the optional features with the related variation points of the architectural models that implement the product line through traceability links. There is wide agreement about using a model that shows, in an explicit and hierarchical way, the variability by means of a feature model in some of their multiple versions like FODA (Kang et al., 1990) or FORM (Kang et al., 1998). FODA features are nodes of a tree, related by various types of edges (Figure 1). The tree root is called the root feature, or concept. The edges are used to decompose this concept into more detailed features. There are AND, X-OR and optional decompositions. Several extensions have been proposed, using directed acyclic graphs instead of simple trees or changing the visual syntax.

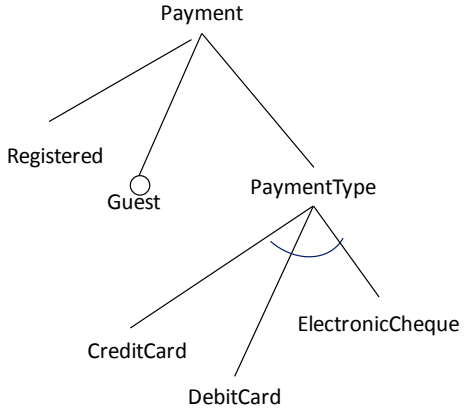


Figure 1: A simple FODA feature diagram.

Laguna M. and González-Baixauli B. (2009).
MOBILE COMMUNICATORS FOR DISABLED PEOPLE.
In *Proceedings of the 4th International Conference on Software and Data Technologies*, pages 5-12

We have also proposed the use of the goal and soft-goal concepts (van Lamsweerde, 2001) for the analysis and definition of the variability in product lines. In fact, we have built an initial prototype that permits the optimum set of features to be selected with respect to the desires of the users, expressed as a set of goals and soft-goals with different priorities (González-Baixauli et al., 2004).

The second aspect of the problem focuses on the connection of the feature model with the design of the solution or product line architecture, usually implemented by an object-oriented framework. This explicit connection allows the automatic instantiation of the domain framework in each specific application. In a previous work (Laguna et al., 2007), we proposed the UML 2 package merge mechanism to orthogonally represent the SPL architectural variations, their relationship with the optional features and finally, using partial class packages, with the structure of the implementation code.

We planned, as a continuation of this work, to test the proposal in realistic situations. Our group has agreements with associations of handicapped people with the aim of developing useful tools for people with several types of disabilities. This background has guided the selection of the application domains. This article is a report of the practical experiences with these techniques in the development of a product line of personalized communicators for people with disabilities, based on mobile devices, typically personal digital assistants (PDA).

A distinctive characteristic is the use of conventional CASE and IDE tools. This is a pre-requisite imposed by the general objective of our approach: to simplify the adoption of the product line paradigm by SMEs. In particular, we have used .NET and MS Visual Studio as development platforms. The personnel involved vary from granted and volunteer postgraduate students to undergraduates finishing their term projects, but they are not specialists in SPL development.

The rest of the article is organized as follows: Section 2 briefly presents the proposed techniques, based on the package merge relationship of UML 2 and the partial class mechanism. Section 3 is devoted to the description of the case study. In Section 4, the related work is analyzed and, finally, Section 5 concludes the article, states some lessons learned and outlines future work.

2 SEAMLESS SPL DEVELOPMENT

Each concrete system in a product line is derived from a complete architecture, selecting or not the optional parts, with respect to the particular functional and non-functional user requirements. This activity is basically a selection process that yields a feature sub-model. This sub-model, through traceability relationships, guides the composition of the code modules. The key aspect of the process is the traceability from features to design and from design to implementation. This traceability is not easily managed for several reasons (Sochos et al., 2004). On the one hand, an optional feature can be related to several elements in a UML model and vice versa. We must therefore assign the traceability relationship between elements of the two levels with a “many-to-many” multiplicity. This fact quickly complicates the global model, making it poorly scalable. The second problem is summarized in the fact that the same basic modeling mechanisms of variability (the specialization in class diagrams or the <<extend>> relationship of the use cases diagrams) are used to express two variability levels: the design of the product line architecture and the design of a specific application that also has variations (for example two valid and alternative payment forms within a sales system).

The solution to this problem has been achieved by modifying or adapting the UML structural and behavioral models, moving from the standard (see the references of the related work Section). In our approach, one of the initial restrictions imposed was to maintain unchanged the UML meta-model, in order to use conventional CASE tools to model the product line. Other obligations were:

a) The technique must allow the location, at one point on the model, of all the variations associated to each optional feature to facilitate the management of the traceability.
b) The technique must separate the SPL from the intrinsic variability of the specific applications.
c) The selected mechanism must have continuity with the implementation models (“seamless development”).

To achieve these objectives, we express the variability of UML models using the package merge mechanism, defined in the UML 2 infrastructure meta-model and used in an exhaustive way in the definition of UML 2 (Object Management Group, 2003).

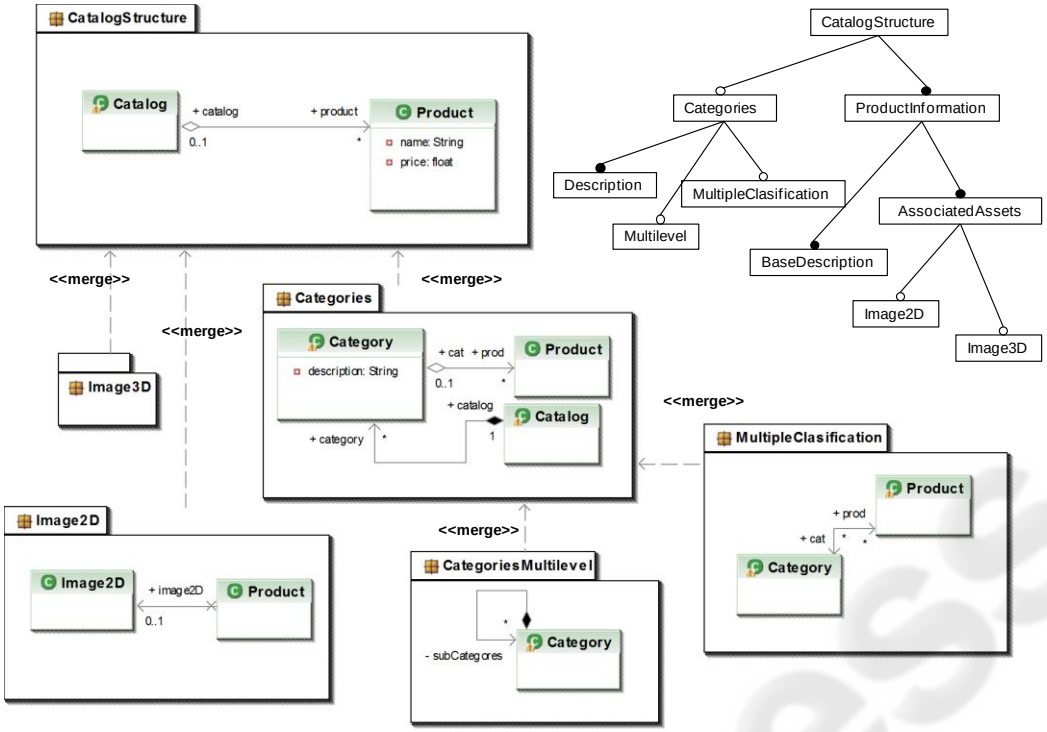


Figure 2: A partial Feature Model and a possible UML design.

The package merge mechanism adds details to the models in an incremental way. The <<merge>>> dependence is defined as a relationship between two packages that indicates that the contents of both are combined. It is very similar to generalization and is used when elements in different packages have the same name and represent the same concept, beginning with a common base. Selecting the desired packages, it is possible to obtain a tailored definition from among all the possible ones. Even though, in this work, we focus on class diagrams, the mechanism can be extended to any UML model, in particular use cases and sequence diagrams (Object Management Group, 2003).

This mechanism permits a clear traceability between feature and UML models to be established. The application to our problem consists in associating a package to each optional feature, so that all the necessary changes in the model remain located (maintaining the UML meta-model unchanged and separating both variability levels).

The package model is hierarchical, reflecting the feature model structure. Considering each pair of related packages recursively, the base package can be included or not in each specific product, but the dependent package can only be included if the base package is also selected. This is exactly how experts decide which features are included or not during the configuration process, and is directly reflected in the final product configuration of packages. Therefore, the application to SPL consists of building the architectural model (including structural –class diagrams-, behavioral -use cases-, and dynamic –interaction diagram- models) starting from a base package that gathers the common SPL aspects. Then, each variability point detected in the feature model originates a package, connected through a <<merge>> relationship with its parent package. These packages will be combined or not, when each product is derived, according to the selected feature configuration. Figure 2 shows an example of application in the e-commerce domain.

Additionally, using partial classes organized in packages, a direct correspondence between design and code can be established. The use of partial classes is a way of managing variability. The aim is to maintain a one-to-one correspondence from features to design and from design to implementation. As an added value, the package structure of the code for each final product of the SPL can be obtained automatically (and passed to the compiler) from the features configuration (Laguna et al., 2007).

Table 1: Comparison of different writing methods.

Writing method	*Speed required*	*Capacity*	*Learning*
Swept	Very slow	Very little	Very little
Sweep (with sound)	Very slow	Very little	Very little
Sweep (groups)	Slow	Very little	Little
Diagonals	Middle	Little	High
Repeated pulsations	Middle	Middle	Middle
Databases	Rapid	Middle	Middle
Traits	Very rapide	High	High
Grouped characters	Rapid	Middle	Middle
Vowels	Rapid	Middle	High

3 CASE STUDY: COMMUNICATORS FOR PEOPLE WITH DISABILITIES

The case study is not very ambitious if we judge it by the number of considered variations but presents interesting problems, due to the constraints imposed by the specificity of mobile device development. The domain analysis has been accomplished starting from the experience with several PDA systems developed in our laboratory. Each one of these originally solved the particular problem of a concrete person with some degree of disability. These systems have been built in collaboration with *Asprona*, a Spanish association that maintains several schools specialized in children with medium/severe disabilities of several types. The main utility of these communicators is that people with different degrees of disability can compose messages using text (selecting the different characters as in a keyboard) or images (that represent different concepts) in a suitable (and usually mobile) device. The suitable methods are compared in Table 1. Once composed, the device can reproduce the message using a text-to-speech conversion (or send it to another device). The product line approach has a clear intention: separate the common parts of these systems from the specialized ones, developing these parts as optional packages. As an immediate result, we have multiplied the number of different available variants.

3.1 Feature Analysis

All the final applications must reproduce the text composed by the user. But, due to the different abilities of the users, it is necessary to consider different writing methods, adapted to each type of disability. For example, if the user is not capable of clicking a button, it is necessary to use a sweeping method. We have considered several (textual and image based) writing methods. Some of them are the

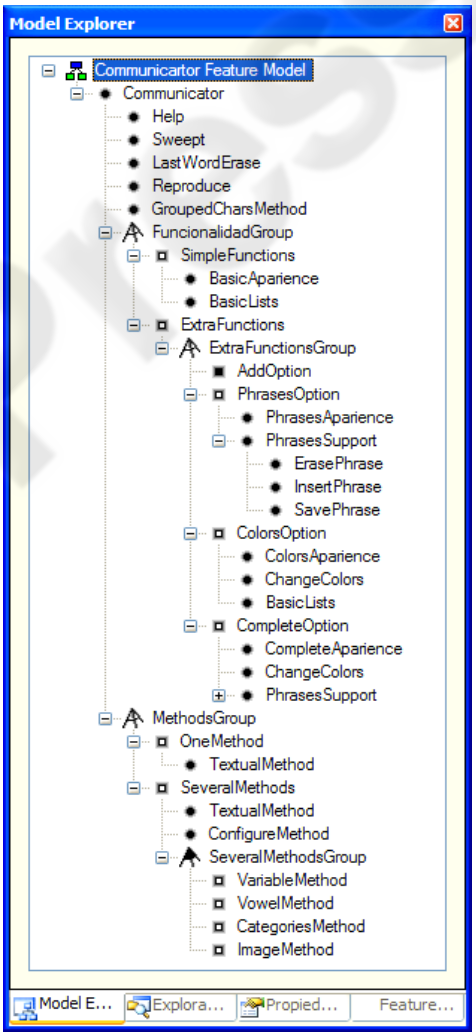


Figure 3: Feature model of the communicator product line.

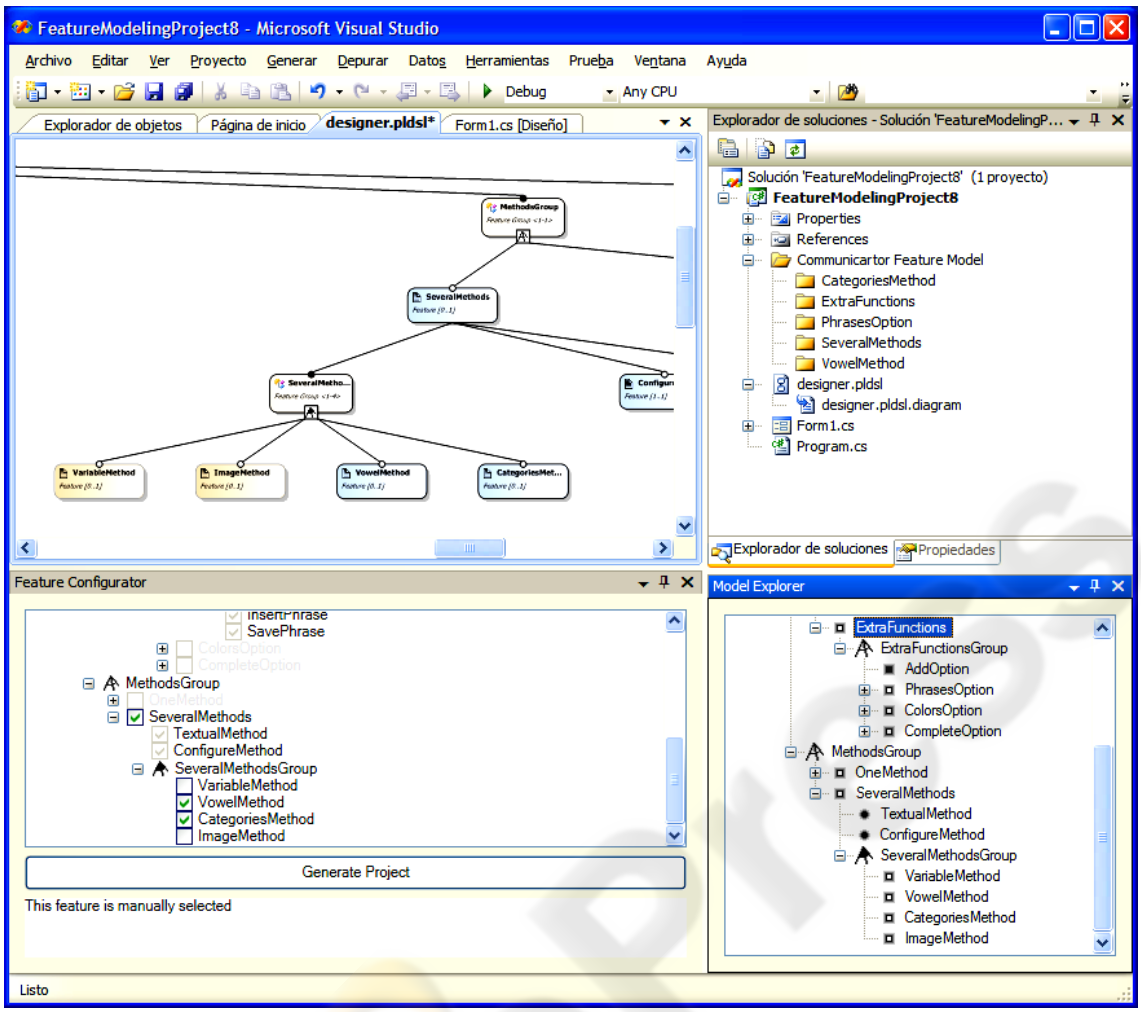


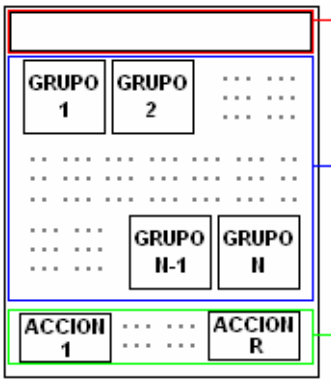
Figure 4: Feature model tool and the communicator product line, solution, configuration and model views.

following:

- Grouped characters method: the main screen shows several groups of characters (AÁBCDE, ÉFGHIÍ, etc.). Selecting a group enables another screen, where the characters of this group appear redistributed, one per cell. The selection of one of them results in that character being added to the text.
- Vowels method: similar to the previous method, but the vowels are represented in independent cells on the main screen, generally reducing the number of pulsations.
- Categories method: the categories of characters (consonants, vowels and numbers) are shown in the initial screen.

Each of the evaluated methods has advantages and inconveniences for people with different degrees and types of disabilities, as shown in Table 1. Using the table as a guide, and adding some complementary aspects such as color management, phrases persistence, etc., the feature model of Figure 3 has been defined.

The feature model has been defined with the Feature Modeling Tool (FMT, available at GIRO site), developed in our group with Microsoft DSL tools as an add-in of Visual Studio. The modeling tool is completed with the
package generation and configuration utilities, as explained in the previous Section. For legibility reasons, the original graphical tree format is depicted in a compact alternative representation (the model explorer, bottom right panel of Figure 4 view of FMT). According to this, each final product can incorporate several writing methods, but all the systems will have at least the grouped characters method. For this reason, the right structure of the feature model has two main alternative branches. If more than a writing method is selected, the exchange



the limitations of the mobile platforms have forced to an *ad-hoc* solution, developing a simple syllabic synthesizer, with the collaboration of the students who lend their voices.

The product line includes eight thoroughly functional applications, compiled from different package combinations (some examples can be appreciated in Figure 6). Pending integration is an optional feature already implemented that will allow wireless and SMS based communication with a desktop computer.

A first working prototype has been delivered to the Asprona association specially configured for a person with speech problems but good manual coordination, as a result of a traffic accident. In this case, the grouped characters method is a good election. The use of the system, fixed to his wheel chair, is helping him to get a greater level of autonomy.

Figure 7: A final prototype, configured using the grouped characters method.

4 RELATED WORK

Though there are many projects that describe variability management mechanisms in terms of requirements and designs, few of them include implementation details. Different authors have proposed explicitly representing the variation points adding annotations or changing the essence of UML. For example, Von der Maßen et al. proposed using new relationships ("option" and "alternative") and the consequent extension of the UML meta-model (Massen & Lichter, 2003). John & Muthig suggest the application of use case templates to represent the variability in product lines, using stereotypes (John & Muthig, 2002), though they do not distinguish between optional variants, alternative or obligatory. On the other hand, Halman and Pohl defend the modification of use case models to orthogonally represent the variation points (using a triangle symbol with different annotations) (Halmans & Pohl, 2003). As for structural models, either the mechanisms of UML are used directly (through the specialization relationship, the association multiplicity, etc.) or the models are explicitly annotated using stereotypes. The work of Gomaa is an example of this approach, since it uses the stereotypes <<kernel>>, <<optional>> and <<variant>> (corresponding to obligatory, optional, and variant classes) (Gomaa, 2000). Similarly, Clauß proposes a set of stereotypes to express the variability in the architecture models: <<optional>>, <<variationPoint>> and <<variant>> stereotypes designate, respectively, optional, variation points (and its sub-classes), and variant classes (Clauß, 2001). Though this type of approximations permits the evolution of the variability to be traced at the different levels, they do not solve the requirement of a one-to-one correspondence between the different models.

Another solution proposed by Czarnecki in (Czarnecki & Antkiewicz, 2005), consists of annotating the UML models with presence conditions, so that each optional feature is reflected in one or, more realistically, several parts of a diagram (perhaps a class, an association, an attribute, etc. or a combination of elements). This technique does not artificially limit the representation of a variant with a unique element and even the color code helps to emphasize the implications of choosing a certain option. However, this visual help is not scalable when more than a dozen variants are handled. In all these approaches, the modification of the UML meta-model (or at least the use of stereotypes) is required.

A completely different approach, focused on implementation instead of requirements or design, is the Feature Oriented Programming (FOP) paradigm (Batory et al., 2004). The optional features are implemented as increments (refinements) in a java-like language. Starting from a base class, these increments are combined using a set of tools, provided with the AHEAD tool suite. Other commercial tools, such as Big-Lever Gears or Pure-Variants offer functionalities. Though these solutions are valid, the learning of new modeling or implementation techniques and the need of specialized CASE and IDE tools represent barriers for the adoption of the approach of product lines in

many organizations; we therefore believe that the solution presented here improves the abovementioned proposals.

5 CONCLUSIONS

In this work the viability of a product line development approach, based on the package merge and partial class mechanisms, has been shown. The use of the proposed mechanisms enables the automated generation of each product from the features configuration. Furthermore, the use of conventional CASE and IDE tools can simplify the adoption of this paradigm, avoiding the necessity of specific tools and techniques as in previous alternatives.

The approach has been successfully applied to the design and implementation of a product line in the domain of communicators for people with disabilities, and implemented with mobile devices.

Current work includes the development of other product lines with industrial or social interest, and the enrichment of the communicator study. In this case, the objective is to evaluate the scalability of the proposal as the optional features increase (which implies an exponential increase in the number of final products). On the other hand, the experience with this type of mobile platform is being used in other domains that combine information capture through PDAs and smart phones with delivery to a central system, configured as a set of Web services. An example of this is a recently launched product line project for monitoring health parameters (such as heart rate, temperature, etc.) in the context of a senior citizen residence, using a combination of wireless sensors and mobile devices. The utility of the product line approach in these domains is evident, as the variety of sensors, parameters, alarm signals, and visualization aspects in the central computer is potentially unlimited.

ACKNOWLEDGEMENTS

This work has been founded by the Junta de Castilla y León (VA-018A07 project) and Spanish MICIINN (TIN2008-05675). We also recognize the collaboration of the *ASPRONA* association, and the students involved in the development of these product lines.

REFERENCES

Batory, D., Sarvela, J., & Rauschmayer, A. (2004). Scaling Step-Wise Refinement. *IEEE TSE* .

Bosch, J. (2000). *Design & Use of Software Architectures. Adopting and Evolving a Product-Line Approach.* Addison-Wesley.

Clauß, M. (2001). Generic modeling using Uml extensions for variability. *Workshop on Domain Specific Visual Languages at OOPSLA.*

Czarnecki, K., & Antkiewicz, M. (2005). Mapping Features to models: a template approach based on superimposed variants. *In proc. of GPCE'05, LNCS 3676*, (pp. 422-437). Springer.

Gomaa, H. (2000). Object Oriented Analysis and Modeling for Families of Systems with UML. *IEEE International Conference for Software Reuse (ICSR6)*, (pp. 89–99).

González-Baixauli, B., Leite, J., & Mylopoulos, J. (2004). Visual Variability Analysis with Goal Models. *Proc. of the RE'2004*, (pp. 198-207).

Halmans, G., & Pohl, K. (2003). Communicating the Variability of a Software-Product Family to Customers. *Journal of Software and Systems Modeling*, 15-36.

John, I., & Muthig, D. (2002). Tailoring Use Cases for product line Modeling. *Proceedings of the International Workshop on Requirements Engineering for product lines 2002 (REPL'02). Technical Report: ALR-2002-033, AVAYA labs.*

Kang, K. C., Kim, S., Lee, J., & Kim, K. (1998). FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures. *Annals of Software Engineering*, 143-168.

Kang, K., Cohen, S., Hess, J., Nowak, W., & Peterson, S. (1990). *Feature-Oriented Domain Analysis (FODA) Feasibility Study.* Technical Report, CMU/SEI-90-TR-21, Software Engineering Institute (Carnegie Mellon), Pittsburgh, PA 15213.

Laguna, M. A., González, B., López, O., & García, F. J. (2003). Introducing Systematic Reuse in Mainstream Software Process . *IEEE Proceedings of EUROMICRO'2003*, (pp. 351-358).

Laguna, M. A., González-Baixauli, B., & Marqués, J. M. (2007). Seamless Development of Software Product Lines: Feature Models to UML Traceability. *GPCE 07.*

Massen, T. v., & Lichter, H. (2003). RequiLine: A Requirements Engineering Tool for Software product lines. *Software Product-Family Engineering, PFE, LNCS 3014 pp* , (pp. 168-180).

Object Management Group. (2003). *Unified modeling language specification version 2.0: Infrastructure. Technical Report ptc/03-09-15.* OMG.

Sochos, P., Philippow, I., & Riebish., M. (2004). Feature-oriented development of software product lines: mapping feature models to the architecture. En *LNCS 3263* (pp. 138-152). Springer.

van Lamsweerde, A. (2001). Goal-Oriented Requirements Engineering: A Guided Tour., (pp. 249-262). Proceedings of the 5 IEEE Int. Symp. on Requirements Engineering.