A Brief Survey on Product Derivation Methodsin Software Product Line

Nesrine LAHIANI
LRDSI Laboratory, C.S. Department, Saad Dahlab University, Blida, Algeria
lahiani.nesrine@gmail.com

Djamal BENNOUAR
LIMPAF Laboratory, C.S. Department, Akli Mohand Oulhadj University, Bouira, Algeria
djamal.bennouar@univ-bouira.dz

Abstract: Product Derivation represents one of the main challenges that a Software Product Line (SPL) faces. Deriving individual products from shared software assets is a time-consuming and an expensive activity. Until now, only few works addressed, in a limited context, a partial evaluation of a reduced number of proposed derivation approaches. The main objective of such studies was the comparison of a proposed approach regarding two or three approaches. The purpose of the study reported in this paper is to set up a framework oriented to evaluate and compare existing SPL derivation approaches. The proposed framework uses a number of criteria which help understanding the capabilities and highlight the strength and the weakness of each SPL derivation approach.

Keywords: Product Derivation; Software Product Line; Application Engineering; Comparison framework

1. Introduction

A software product line (SPL) is as a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way [1]. In a software product line context, software products are developed in two processes, i.e. a domain engineering process and an application engineering process as graphically figure 1 represents the general SPL engineering process, as it can be found in the research literature the SPL approach makes a distinction between domain engineering where a product is derived based on the platform components and application engineering where individual products using the platform artefacts are constructed [2]. The process of creating these individual products from a product line of software assets is known as product derivation [3].

Product derivation is a key activity in application engineering. It addresses the construction of a concrete product from the product line core assets. The term product derivation refers to the complete process of constructing a product from product family software assets [3].

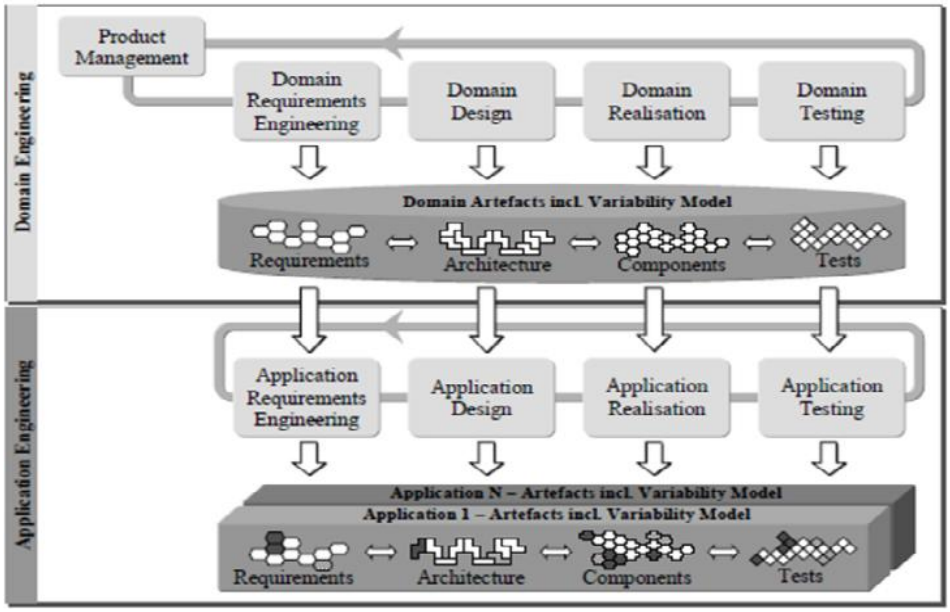

Figure 1. The software product line engineering framework **[3].**

A number of SPL derivation approaches and tools have been proposed in the last decade, e.g. PuLSE-I-[4], DREAM [5], KobrA[6], COVAMOF[7], Pro-PD [8] and DOPLER^{UCon} [9]. Until now only a few and limited work has been conducted to evaluate and classify existing derivation approaches. The evaluation of existing approaches is very important in order to understand and appreciate their true capability, strength and weakness regarding a number of criteria, their actual goals and their readiness to be reused in the process of building a new SPL. Setting up a framework for classifying, evaluating, and comparing existing SPL derivation approaches represents the main purpose of the study reported in this paper.

In this paper we focus on one domain scope, which is individual product line. In this domain, single product is used to derive several related products

The remainder of this paper is structured as follows: Section 2 provides background knowledge on product derivation. In Section 3 we introduce a comparative framework for evaluating product derivation methods. Then, the five product derivation methods are briefly presented in Section 4 and compared against the framework in Section 5. The most remarkable observations of the comparison close the paper.

2. Product Derivation in Software Product Line

Rather than describing a single software system, the model of a software product line (SPL) describes the set of products in the same domain. This is done by distinguishing elements shared by all the products of the line, and elements that may vary from one product to another.

The ultimate goal of a product derivation process, as part of the SPL application engineering lifecycle, is to produce a configurable or configured software product. However, product configuration can be enacted at the beginning of the derivation process at early binding times, or it can be also executed at a very late stage if a product has to be reconfigured once deployed. Configuration is sometimes done to select the variable options that will be included in a product before the variability is realized to concrete values, while in other cases, a reconfiguration process happens at the end of the product line or during system execution. Moreover, product

configuration and variability realization can also overlap at the same binding time if we realize the variants at the same time these are selected. At the end of the derivation process, products are installed and deployed in the physical nodes of the system [10].

Software products are developed, in the context of product line engineering, according to two separate processes, namely domain engineering and application engineering. The former is dedicated to core asset development while the latter is aimed at yielding products.

We focus in this paper at application engineering known also as product derivation (PD). PD has been defined in many different ways, in [11] John D. McGregor defines it by "Product derivation is the focus of a software product line organization and its exact form contributes heavily to the achievement of targeted goals".

Deelstra, Sinnema, and Bosch define product derivation by, "A product is sa.id to be derived from a product family if it is developed using shared product family artifacts. The term product derivation therefore refers to the complete process of constructing a product from product family software assets" [3].

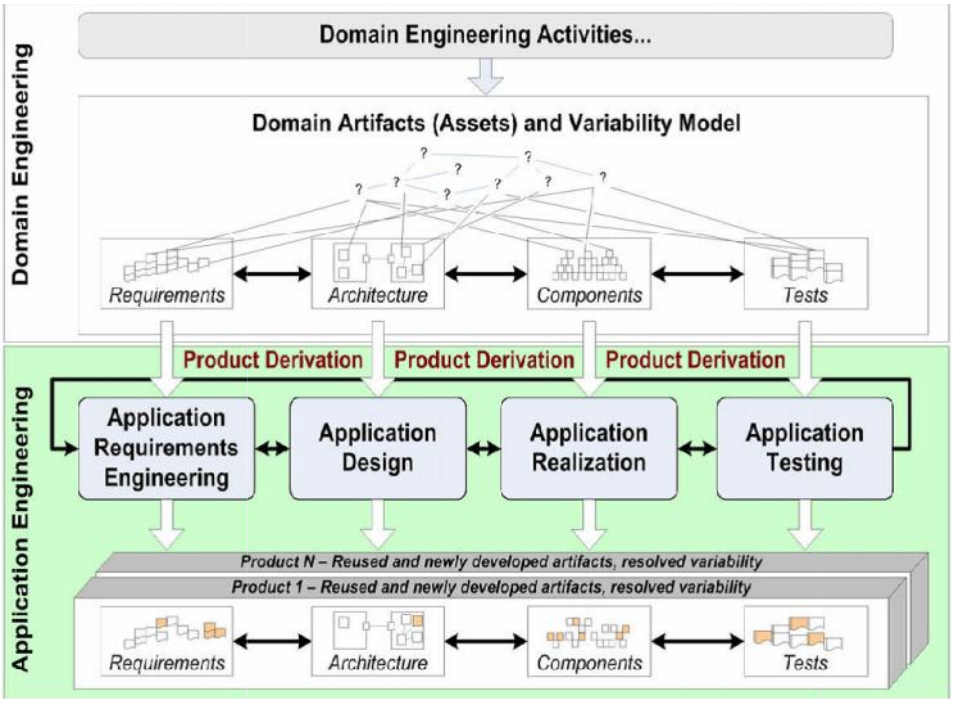

Figure 2.SPLE processes.The upper white vertical arrows represent the product derivation process of selecting and customizing reusable assets during application engineering [12]

Figure 2 depicts a high-level application engineering process. The upper white vertical arrows represent the product derivation process of selecting and customizing reusable assets during application engineering. The lower white arrows indicate deployment activities necessary to arrive at a final product (e.g., deploying and integrating new components developed to address customer requirements with the existing derived components) [12].

3. An Evaluation Framework

To evaluate each approach we identify a set of criteria presented in Table 1. We start the evaluation from the element of the problem situation, i.e. the approach context (input requirement and output product of the method). The second category is the problem solving process, i.e. what does the method itself contain (process, artifacts, tool...). The last category brings the two previous categories together through self-evaluation to evaluate the method output.

The goal of our evaluation is to provide an overview of current product derivation in SPL and find out if - and how - the methods differ from their each other. With various questions this study tries to address, e.g. maturity, practicality and scope of the methods to find differences. These elements are considered in the category of 'contents' by questioning if the methods satisfy the definition of a method. The elements of the framework are refined to cover features special for product line methods (e.g. modeling variability).

We briefly define each element framework next:

- ***Requirement specification*** customer's requirements must be specified before using it as input, the specification could be textual, model,…
- ***Final Product*** the product at the end of the process could be an executable application, a hardware, …
- ***Method Process*** the activities that must be followed to derive a product that meets customer's requirements.
- ***Artifacts*** During the derivation process each method produces a set of artifacts (reference architecture, test, documents...).
- ***Tool*** used some sub-process need tools as eclipse or to model UML.
- ***Modelling variability*** is to efficiently describe more than one variant of a system. Variability can be expressed in stand-alone models, such as feature diagrams.
- ***Method maturity*** validates a method by a case study that could be industrial or academic show how the method is mature or not.

A more complete summary of this section is given in Table 1.

Table 1: The categories and the framework elements for Characterization and Comparison of product derivation methods

Framework element	Description
Requirement specification	What is the trigger of the approach?
Final Product	What is expected at the end?
Method Process	What are the activities needed to derive a product?
Artifacts	What are the artifacts created and used during derivation process?
Tool used	What are the tools supporting the method?
Modeling variability	What kind of model does the approach use to modeling variability?
Method maturity	Has the method been validated in practical industrial cases?

The next section presents the six product derivation methods and how this framework can be used to compare each method against a criterion.

4. Overview of Derivation Method

We first provide a brief overview of the six selected approaches and then show the results of our analysis and comparison in section 6 based on the adapted evaluation framework introduced in Section 4.

5.1. PuLSE-I

PulSE (Product Line Software Engineering) is a full life-cycle product line [14]; it is centered on three main elements: the deployment phase, the technical components and the support components.

The PuLSE-I is the application engineering process of PuLSE, it is centered on the instantiation of the product line infrastructure (the I in PuLSE-I stands for instantiation) [4].

PuLSE-I details how a single product can be built efficiently from the reusable product line infrastructure built during the other PuLSE activities. The trigger of PuLSE-I is a customer or the management having a product request that can be satisfied by the product line (i.e., the requested product is potentially in the scope of the product line).

The process is started by the creation of a plan for the development of the specific product; this plan is based on the product request and the scope definition. The next step is to derive from the domain decision model a product line model, after that an instantiation of the reference architecture is required. In the next step the product is assembled from assets, but before the delivery of the final product an acceptance test is performed.

Method Process. The method is divided into fourmain areas of activity, or phases:

Plan for the product line instance (the product): Determine whether all characteristics of the required product are covered by the product line

Create project plan: Define what is product-specific and what can be fulfilled by the product line

Instantiate and validate product line model: Incrementally resolve decisions defined in the product line model (representing variation points)

Instantiate and validate reference architecture: Instantiate variability to derive an ‘‘intermediate architecture’’ from the product line, validate, and thenmodify if necessary

Product construction: Lower level design, implementation, and testing based on reusable assets

5.2. DREAM

In [5] Kim et al. proposed a methodology called DREAM a practical product line methodology stands for DRamatically Effective Application development Methodology. DREAM adopts the key activities of SPL and model transformation feature of MDA. The methodology allows semi-automatically development of a large number of applications that vary on behavior and implementation platform.

Method Process. The process consists of 9 phases, and each phase produced various artifacts with different characteristics.The derivation process starts after three phases of framework engineering; it used the model created during framework engineering.

- Application Requirement Analysis
- Application Specific Design
- Framework Instantiation
- Model Integration
- Application Detailed Design
- Application Implementation

5.3. KobrA

The KobrA method stands for Komponentenbasierte Anwendungsentwicklung that is German for “component-based application development” [6]. The approach has been developed by Fraunhofer Institute for Experimental Software Engineering (IESE) [14].

KobrA integrates two approaches component-based and software product line into a unified approach. It designed for modeling architecture, for developing both single and family system and to support a model driven architecture (MDA).

KobrA and PuLSE have a relationship between their activities which made KobrA an object-oriented customization of the PuLSE method. KobrA has two main activities: first framework engineering create, and later maintain, a generic framework that embodies all product variants that make up the family, including information about their common and disjoint features. Second the application engineering instantiate this framework to create particular variants in the product family, the process defined a decision model to instantiate a concrete application to the customer, based on component-based product line.

Method Process. Application engineering uses the framework built during framework engineeringto construct specific applications in the domain covered by the framework.The application engineering process is split into two primary steps:

- **Context realization instantiation:**It starts when the software development organization has established an initial contact to a potential customer who is interested in a software system in the domain of one of the organization's frameworks. The outputs of this process are the context decisions and a concrete realization of the application's context.

- **Framework instantiation:**It starts when the application context realization is (partially)created and thus also the context decisions (partially) exist.The context decisions are used to initially instantiate the generic

5.4. COVAMOF

COVAMOF (Configuration in Industrial Product Families VAriability Modeling Framework) [15] an approach developed by Deelstra et al. In [2] Deelstra et al. provides a framework of terminology and concepts for product **derivation, and also presents a generic derivation process that consists of two phases: an initial and iterative** phase.

COVAMOF is supported by a tool-suite, called COVAMOF- VS This tool-suite is implemented as a combination of Add-Ins for Microsoft Visual Studio .NET. It is designed for creating variability models of a product family, and using these models for configuration of individual products. It provides variation point and dependency views on variability models and allows defining, configuring, and realizing products following the COVAMOF derivation process.

Method Process. The COVAMOF derivation process is an instance of the generic derivation process except it does not focus on the difference between the initial and iterative phase, but it divided the process into four main steps as presented in [16]

Product definition: Defining customer and product name

Product configuration: Binding of variation points based on customer requirements.

Product realisation: Tool-based translation of the configuration of the variability model to a configuration of an executable product.

Product testing: Determining whether the product meets the customer requirements and deciding whether an additional iteration (product configuration/realisation/testing) is required.

5.5. Pro-PD

Through a series of research phases using sources in industry and academia, O'Leary et al. has developed a process reference model for product derivation (Pro-PD) [8] in Lero (the Irish Software Engineering Research Center) with a specific goal defining a process reference model for product derivation as a foundation for situation-specific process approaches to product derivation.

Method Process. Pro-Pd is structured around five essential activities: initiate project, identify and refine requirements, derive the product, develop the product, test the product, and management and assessment. Each of these activities contains roles, tasks and artefacts used to derive products from a software product line. Tasks are units of works that consume or produce one or more products, Pro PD groups related tasks into activities, each activity has a specific goal, inputs and outputs. These tasks roles are assigned to human activity

5.6. DOPLER^{UCon}

DOPLER^{UCon} (Decision-Oriented Product Line Engineering for effective Reuse: User-centered Configuration) has been developed by Mag. Rick Rabiser at Christian Doppler Laboratory for Automated Software Engineering [9]. It is a tool-supported approach for product configuration with capabilities for adapting and augmenting variability models to guide sales people and application engineers through product derivation. Particular emphasis lies on support for

requirements acquisition and management. The approach also aims to make variability models accessible to "non-technicians" such as sales people or customers to fully exploit the benefits of PLE.
Method Process. DOPLER^{UCon} process contains six different activities: (1) Domain expert prepare product configuration by creating the derivation model. (2) Users defined in the derivation model perform the actual product configuration by taking decisions visible to them. (3) In parallel, they capture arising product-specific requirements. (4) Based on these requirements developers conduct additional development. (5) Finally, engineers integrate new developments with the selected and customized product and deploy it for the customer [9].

5. Discussion

The purpose of this discussion is to offer guidelines related to the selection of the most suitable method for product derivation. We selected six approaches for a product derivation in software product line, not to rate which one is the best but to characterize, to compare and to see how they differentiate and resemble. The purpose of this discussion is to analyse the results obtained.
We summarize the results of our analysis and comparison of PuLSE-I, DREAM, KobrA, Pro-PD, and DOPLER^{UCon} using the evaluation framework first introduced in Section 3 as it is shown in Table 2.
Some observations are evident first the context of the approaches; for specific goal, we can identify in all approaches a collective goal, which is “satisfy customer’s needs”. Then same for the input we can observe that to start the process each method needs “customer’s requirements” which is almost the same for all methods, it’s differ in the format e.g. Pro-PD translates this requirement before using it. Likewise the output of each approach is almost the same which is the final product that meets customer's requirements.
Secondly the contents of the approaches, at this stage we can see how the approaches differ even though the goal is the same, but no method is similar to another. Some are abstract and difficult to apply.
When commencing the research, we identified some limitations to current approaches; e.g. dream that did not give any guidance or process support, moreover any details about tools used during the process. However Kim et al. do not give any details about models used during phases nor the transformation tool. Moreover a high level description of the activities and no guidance is provided on how the approach could be applied
Same for KobrA, which does not provide does not provide a process support, neither a support for the development of a specific product.
The last criterion was validation, Pulse-I and Dream do not give any details about how they validate their processes. For the others, they have been validated in practical industrial case studies, academic or both as DOPLER^{UCon} did.
Finally, in practice how to know which method is the most suitable this is not the purpose of this paper, but to select the right one each criterion must fits with problem context.

Table 2 Analysis and comparison of PuLSE-I, DREAM, KobrA, Pro-PD, and DOPLER^{UCon}

Approaches	Framework Element
How does the method specify customer's requirements?	
PuLSE-I-	A detailed project plan that considers the set of characteristics upon which the customer (or the marketing) and the developers have agreed. Each required characteristic is specified by a detailed description of how it will be supported.
DREAM	Translated customer requirement created based on customer requirements and a glossary.
KobrA	Decision models are used to capture the variabilities within applications in the product line.
COVAMOF	The engineer creates a new Product entity in the variability model. The properties of Product entities are the customer, a unique name for the product, and variation points that have been bound for the product
Pro-PD	Translate the Customer Requirements into the internal organisational language. The Product Analyst used a customer terminology Glossary
DOPLER^{UCon}	Domain experts create a derivation model which can have a name, description, and purpose. It is further on used to present decisions in the variability model to decision-makers during product configuration, to store their decisions, and to capture product-specific requirements.
What are we expected at the end?	
PuLSE-I-	The delivery process depends on the market size, For the mass market, the system must first be packaged together with an installation guide to enable any customer to install it at his/her machine. For small markets, like for individual systems for single customers, the system is installed by the developers at the customer's site.
DREAM	Produce executable application code and associated implementations.
KobrA	The final results are the application decisions consisting of the context decisions and the Komponent hierarchy decisions, together with the application realization and the application tree.
COVAMOF	A configuration of an executable product that satisfies all customer requirements.
Pro-PD	Product Release that satisfy all customer requirement.
DOPLER^{UCon}	Product delivered and/or to be installed for the customer
What are the artifacts created and used during derivation process?	
PuLSE-I-	Project plan Domain decision model instance Architecture decision model instance Low level configuration Code Test results
DREAM	Application Analysis Model Application Specific Design Instantiated Framework Integrated Application Model Detailed Design Model Application Code
KobrA	Context decisions Instance of generic Komponent hierarchy

COVAMOF	Product entity XML-based feature models C++/C# source files
Pro-PD	Two different kind of artifact: Software artifact. Documentation artifact.
DOPLER^{UCon}	Derivation model Code source files Specific–asset
What are the tools supporting the method?	
PuLSE-I-	**DIVERSITY/CDA** visualizes the impact of decision made which enable an immediate validation.
DREAM	Not reported
KobrA	Commercial UML tool
COVAMOF	**COVAMOF-VS** provides two main graphical views, i.e. the variation point view, and the dependency.
Pro-PD	Eclipse Process Framework(EPF) researcher used it to model the derivation product process and create a formalised version of Pro-PD
DOPLER^{UCon}	**ProjectKing** The configuration preparation activity of DOPLER^{UCon} is supported by the tool ProjectKing. Domain experts like project and sales managers use the tool to create derivation models based on variability models. **ConfigurationWizard** Domain experts and (software) engineers use the tool to review and communicate available variability, resolve variability, customize assets by taking decisions, generate configurations, capture product-specific requirements, relate product-specific requirements to the existing variability, and generate requirements specifications
What kind of model does the approach use to modeling variability?	
PuLSE-I-	**PuLSE-CDA** (Customizable Domain Analysis)[6].
DREAM	Not specified. Any reasonable representationscheme may be used.
KobrA	**UML**in KobrA, each Komponent (KobrA component) in the framework is described by a suite of UML diagrams as if it were an independent system in its own right [17].
COVAMOF	**COVAMOF** a variability modeling framework that models variability in terms of Variation Points and Dependencies [11].
Pro-PD	Not reported
DOPLER^{UCon}	**DOPLER^{VM}**the approach was influenced by earlier work by [18] and the Synthesis project [19] and comprises two main elements: asset and decision.
Has the method been validated in practical industrial cases?	
PuLSE-I-	The PuLSE approach has been applied in case studies, for example [20] and [10]
DREAM	Not reported
KobrA	The domain of Enterprise Resource Planning. It used in the development of the KobrA workbench.
COVAMOF	• Deelstra, S., Sinnema, M., Bosch, J., 2005. Product derivation in software product families: a case study [3]

	• Industrial validation of COVAMOF. J. Syst. Software [21].
Pro-PD	• Industrial case study : ROBERT BOSCH GMBH • further validation of Pro-PD through an inter-model evaluation with the SEI Product Line Practice Framework
DOPLER^{UCon}	• Siemens VAI : The case study focuses on SVAI's automation software for the continuous casting technology. • Business Software BMD.

6. Conclusions and Future Work

Product Derivation represents one of the main challenges that a Software Product Line (SPL) faces. Deriving individual products from shared software assets is a time-consuming and an expensive activity.

In this paper presented a framework that focused on comparing and evaluating different product derivation approaches. We have compared six different methods: Pulse-I, Dream, KobrA, COVAMOF, PRO-PD and DOPLER^{UCon} against a set of developed criteria (Table 1).The aim of this comparison frame work is not to rate which one is the best but to characterize, to compare and to see how they differentiate and resemble.

We believe that our work can lay some ground for further research activities. For instance as pointed out in [3] "deriving individual products from shared software assets is a time-consuming and expensive activity" We hope that this paper can thus provide an initial step towards a better understanding of the similarities and differences of the six approaches.

As future work, we plane to classify the approaches according to the derivation techniques. Also, we intend to improve the framework by adding more criteria and more current product derivation approach.

References

1. Clements,P., Northrop,L. Software Product Lines: Practices and Patterns. The SEI series in software engineering. Addison-Wesley, Boston, 2002.
2. Hotz, L., A. Gunter, and T. Krebs, A Knowledge-based Product Derivation Process and some Ideas how to Integrate Product Development, in Proc. of Software Variability Management Workshop. 2003: Groningen,TheNetherlands.
3. Deelstra, S., Sinnema, M., and Bosch, J., Product derivation in software product families: a case study. The Journal of Systems and Software, 74:173–194, 2005.
4. Bayer, J., Gacek, C., Muthig, D., and Widen, T. PuLSE-I: deriving instances from a product line infrastructure. In: 7th IEEE International Conference and Workshop on the Engineering of Computer Based Systems, pp. 237-245.2000.
5. Kim, S.D., Min, H.G., Her, J.S., and Chang, S.H., DREAM: A Practical Product Line Engineering using Model Driven Architecture, in Proceedings of the Third International Conference on Information Technology and Applications (ICITA"05)., IEEE Computer Society: Washington, DC, USA. p. 70-75. 2005.
6. Atkinson, C. Component-based product line engineering with UML. Pearson Education2002..
7. Sinnema, M. and Deelstra, S., Industrial Validation of COVAMOF. Journal of Systems and Software, 2008. 81(4): p. 584-600.doi:10.1016/j.jss.2007.06.002

8. O'Leary, P., 2010. Towards a Product Derivation Process Reference Model for Software Product Line Organisations. PhD Thesis, Department of Computer Science and Information Systems. University of Limerick, Limerick,2010 p. 277.
9. Rabiser, R. A User-Centered Approach to Product Configuration in Software Product Line Engineering, in Christian Doppler Laboratory for Automated Software Engineering, PhD Thesis, Institute for Systems Engineering and Automation, Johannes Kepler University, Linz, 2009.
10. Capilla, R., Bosch, J., & Kang, K. C. (2013). Systems and Software Variability Management. Concepts Tools and Experiences. t, DOI 10.1007/978-3-642-36583-6_6, © Springer-Verlag Berlin Heidelberg 2013
11. John D. McGregor .Goal-driven Product Derivation. Clemson University and Luminary Software LLC, U.S.A. Journal of object technology.2009
12. Rabiser, R., Grünbacher, P. & Dhungana, D., Requirements for product derivation support : Results from a systematic literature review and an expert survey. Information and Software Technology, 52(3), pp.324–346. Available at: http://dx.doi.org/10.1016/j.infsof.2009.11.001.
13. Bayer, J., Flege, O., Knauber, P., Laqua, R., Muthig, D., Schmid, K., Widen, T., DeBaud, J.M. "PuLSE: A methodology to develop software product lines." In Proceedings of the Symposium on Software Reusability (SSR'99), May 1999.
14. Atkinson, C., Bayer, J., and Muthig, D., Component-based product line development: the KobrA approach, in Proceedings of the first conference on Software product lines : experience and research directions. 2000,
15. Sinnema, M., Deelstra, S., Nijhuis, J., and Bosch, J., COVAMOF: A Framework for Modeling Variability in Software Product Families. In: Proc. 3rd Int'l Conf. Software Product Lines (SPLC 04), San Diego,.2004
16. Sinnema M, Deelstra S, Hoekstra P. The COVAMOF derivation process. In Morisio M, editor, REUSE OF OFF-THE-SHELF COMPONENTS, PROCEEDINGS. BERLIN: Springer-Verlag. 2006. p. 101-114. (LECTURE NOTES IN COMPUTER SCIENCE).
17. Atkinson,C. Baye,J. Laitenberger,O. and Zette,J. Component-Based Software Engineering: The KobrA Approach. Fraunhofer IESE Sauerwiesen 6D-67661 Kaiserslautern, Germany.2000
18. Schmid,K. and John,I.. *A Customizable Approach to Full-Life Cycle Variability Management*. Journal of the Science of Computer Programming, Special Issue on Variability Management, 53(3), pp. 259-284. 2004.
19. Software Productivity Consortium, 1991 Software Productivity Consortium. *Synthesis Guidebook*. SPC-91122-MC. Herndon, Virginia: Software Productivity
20. Weiss, D., Lai, C.., and, Tau R., Software product-line engineering: a family-based software development process. Addison-Wesley, Reading, MA, 1999.
21. Sinnema, M. and Deelstra, S., Industrial Validation of COVAMOF. Journal of Systems and Software, 2008. 81(4): p. 584-600.