Framework for Evolving Software Product Line

Sami OUALI, Naoufel KRAIEM, Henda BEN GHEZALA

RIADI Lab, ENSI, Compus of Manouba
Manouba, Tunisia

samiouali@gmail.com, naoufel.kraiem@ensi.rnu.tn,
henda.bg@cck.rnu.tn

Abstract. *Software product line engineering is an approach that develops and maintains families of products while taking advantage of their common aspects and predicted variabilities. Indeed, software product lines (SPL) are an important means for implementing software variability which is the ability of a system to be efficiently extended, changed, customized or configured for use in a particular context. Variability needs in software are constantly increasing because variability moves from mechanics and hardware to software and design decisions are delayed as long as economically feasible. Numerous SPL construction approaches are proposed. Different in nature, these approaches have nevertheless some common disadvantages. We have proceeded to an in-depth analysis of existing approaches for the construction of Software Product Line within a comparison framework in order to identify their drawbacks. We suggest overcoming these drawbacks by an improvement of the tool support for these approaches and for their interactivity with their users. We propose to study a particular software product line which is ERP as experimentation.*

Key words: Software Product Line, variability, comparison framework, ERP.

1 Introduction

The Software Engineering Institute (SEI) defines the software Product Line as [1], "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". Software product line engineering is an approach that develops and maintains families of products while taking advantage of their common aspects and predicted variability's [2]. The Software product line (SPL) is one of the important means for implementing software variability. Indeed, variability is the ability of a system to be efficiently extended, changed, customized or configured for use in a particular context [3]. Variability needs in software are constantly increasing. Indeed, the variability is moving from mechanics and hardware to software. Currently, design decisions are delayed as long as economically feasible.

SPL engineering is considered as unavoidable approaches to support reuse in systems development, as a viable and important software development paradigm. It allows to companies to realize a real improvements in time to market, cost, productivity, quality and flexibility. In fact, SPL techniques are explicitly capitalizing on commonality. They try formally to manage the variations of the products in the product line.

To understand the problem of variability in software product lines, we've studied a number of references to provide an overview of the current activities in this area. The reference [28] presents a model for product line variability. They try to model variability by feature model, which describes functional and nonfunctional requirements of the product line. This feature model structures the common and variable features. The architectural variability model is described by architectural variation points. The reference [3] presents the variability as the key to software reuse and he asserts that variability can be associated with different abstraction levels of development (from requirement specification to running code). They propose a

DOI : 10.5121/ijsea.2011.2204

terminology for the description of variability in terms of variations and variants using features as a useful abstraction for describing variability. The reference [29] has characterized and compared different software product line approaches to aid in the selection of the best approach and variation mechanisms depending on context.

In the literature, the majority of variability research concerns requirements and architecture. But some works deals with implementation, verification and validation and software product line management. There are many SPL construction methods [3] [4] [5] [6] [7] [8] [9]… In this article, we present four of these methods. We have tried in our choice to present a diversity of methods which deal with SPL in different way.

Despite their diversity, different SPL construction methods have some common drawbacks. To identify them, we elaborate a comparison framework. From the application of our comparison framework on the four selected methods, we realize that we have a lack of sufficient tool support for them and for their interactivity with their users. Moreover, some of the proposed methods are using proprietary notations which can handle some problems of standardization and interoperability…

Thus, this study joins the SPL engineering field with the proposal of a framework used for comparing different construction method, for identifying their drawbacks, and suggesting some ideas to solve them.

Studying software product line in a concrete domain as ERP can be useful for the proposal of new method for the construction of software product line. Software Product Line fits to ERP business. ERP systems can benefit greatly from the concepts of commonalities and variabilities to enhance evolutionability and maintainability.

This paper is organized as follows. A brief description is presented of different concept concerning software product line and variability in the next section. Our comparison framework is described in section 3 and it is applied on four selected SPL's construction methods in the fourth section. In section 5 we present our experimentation. The section 6 concludes this work with our contribution and research perspectives.

2 Software product line and variability concepts

Product line engineering has become an important and widely used approach for the efficient development of whole portfolios of software products [23]. This approach is based on the undertaking of the development of a set of products as a single, coherent development activity. Indeed, products are built from a collection of artifacts from a core asset base that have been specifically designed for use. This approach aims to enable order-of-magnitude improvements in quality, time to market, cost, and productivity, compared to one-at-a-time software system development [22]. Core assets include not only the architecture and its documentation but also specifications, software components, tools such as component or application generators, performance models, schedules, budgets, test plans, test cases, work plans, and process descriptions [22].

Organizing software development according to the product-line paradigm has many implications on the overall development activity. Therefore it is important to achieve a clear understanding of some basic concepts concerning software product line and variability.

2.1 Software Product Line

The definition of the fundamental term of software product line has evolved from a definition as conventional reuse to a derivation from a single common product. The term software product family is proposed in 1976 by David L. Parnas and defined it as:"A set of programs constitutes a [software] product line whenever it is worthwhile to study programs from the set by first studying the common properties of the set and then determining the special properties of the individual members." [24] This mixture of commonality and variability is one

of the important features of a software product-line. The degree of commonality is necessary in order to be able to apply software product line development methods. When the products are too different from each other, the overhead to describe them as members of a single product line is too high to gain substantial benefit from this approach. Clements and Northrop define the term software product line as "A set of software products ... that are developed from a common set of core assets in a prescribed way." [22]

Currently, the products of software product line are derived from a single common product definition and no longer developed from the composition of reusable parts. In this single product definition, we can find a description of the way that products differ from this definition. This idea is illustrated in (Fig 1.c) where we can see that only variable system description is defined. The illustration (Fig 1) presents also the difference between the adoption of software product line approach and conventional reuse approach. In case of conventional reuse, we should define a complete system description for each product (Fig 1.b).

SPL engineering is defined in the literature [12] by distinguishing two levels of engineering: Domain Engineering and Application Engineering. The domain engineering concerns the development and evolution of the product line infrastructure. The application engineering concerns the definition of individual product instances to be derived from that infrastructure.

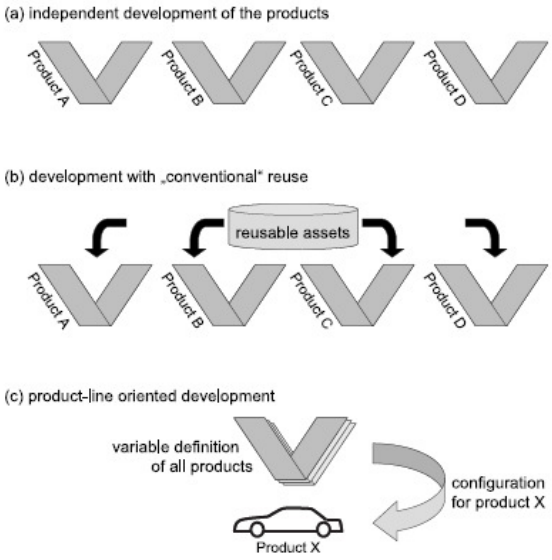

Fig. 1. Difference between Software Product Line approach and conventional reuse. This shows that in conventional reuse we define a complete system description for each product but in product line approach a single common product is defined.

Software product line engineering allows the shifting of the focus of development and evolution from the individual products to the entire product line. Indeed, the product line becomes a first-order entity of development. The advantage of this approach is that the relations between products, especially their commonalities and variabilities, become concrete entities for the development and evolution. This benefit is important for strategic product line scope, adaptation of market needs and reuse of development artifacts.

2.2 Variability

Variability is the ability of a system to be efficiently extended, changed, customized or configured for use in a particular context [3]. Another definition presents variability as the ability of a system, an asset, or a development environment to support the production of a set of artifacts that differ from each other in a preplanned fashion [27]. In this definition variability means the ability of a core asset to adapt to usages in the different product contexts that are within the product line scope. Indeed, variations in a product line context must be anticipated.

A core asset is an artifact or resource built to be used in the production of some products in a software product line [27]. Core assets can be software components, architecture and documentation, analysis models, configuration management plans, interface specifications... Anything used in the creation of a product in software product line is considered as core asset. Product developers can create product assets from use core assets by selection and modification or by creation. We mean by the creation with selection and modification that a core asset is selected and modified or configured to meet the real need of the product to build. The resulting product asset is considered as a variant of the original core asset. With creation, a core asset is used to create a totally different product asset.

A variable part is the position in an asset where a variation can took place. Everything else is a common part. Common parts will not change as the core asset is used from product to product. When a core asset is created, the common part is produced. For the variable part, they can be expressed as alternatives or empty. The realization of a variable part is called a variant. This realization is the result of exercising the variation mechanisms. The inclusion of a core asset in a product is under the specification of some conditions. These conditions are the guaranty that the inclusion is possible. In this case, we try to adapt the variable part of the asset regarding this condition which depends on the required product configuration description.

2.3 Modeling Software Product Lines

The purpose of Variability modeling is to present an overview of a product line's commonality and variability. Variability modeling terms concerns also commonality modeling. In some works the term commonality / variability modeling is preferred. Variability modeling tries to achieve its goals by way of abstraction. This means that some aspects of the product line's variability are intentionally left out of consideration. This aims to reduce the amount of information to a manageable level and to put stress on the important aspects while hiding unnecessary detail. The content of a variability model serves as a basis for defining variability within the artifacts that make up the product-line infrastructure as well as for configuring individual product instances and deriving them from the infrastructure.

With traditional software development, we need to model a single software product. In this case, the construction of the delivered product concerns the manipulation of some development artifacts which can be grouped into the different phases of software life cycle (analysis, design, implementation and verification & validation). Fig.2 illustrates the development artifact in traditional software development.

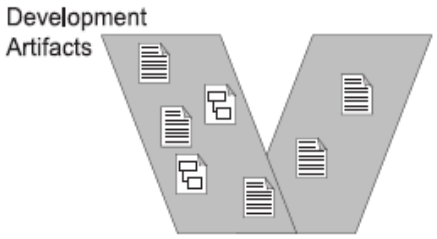

Fig. 2. Traditional software development. This shows the development artifact in traditional software development.

In software product line development context, the purpose is to develop not only a single product but several more or less distinct ones. These products are developed together. Information captured in the development artifacts concerns all the product of the software product line and not only each product separately. In this case, variability may occur within each development artifact. Indeed, some artifact can be needed in only some products, thus their content becomes variable by the introducing of variation point.

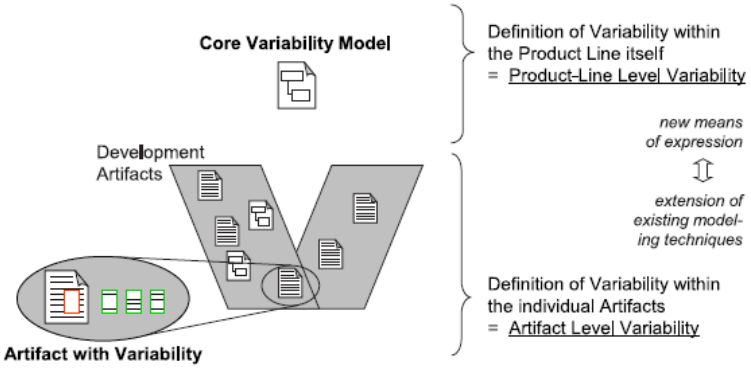

Fig. 3. Software product line development. This shows the development artifact in software product line approach development.

2.4 Variation Mechanisms

A variation mechanism can be introduced into a variant component to take advantage of their similarities. Variation mechanism allows developer to keep a single component which can be adapted in case of need with a certain degree of variation. As known, we have common part and variable part in a component to build for software product line. The component developer must choose the appropriate variation mechanism to encapsulate the variable part. To make a decision about what variation mechanism to use depends on its impact on quality (performance needed or memory consumption) and its cost requisite for the use and implementation. As variation mechanism, we can mention inheritance (Object-oriented languages), component substitution, plugins (framework programming), templates, parameters, aspects (aspect-oriented programming), runtime conditionals…Taxonomy of variation mechanisms can be found in many works [25] [26].

2.5 Software product line approaches

In the literature, the majority of variability research concerns requirements and architecture. But some works deals with implementation, verification and validation and software product line management. For the requirement, a large works concerns the variability modeling in feature models, which represents the main requirements artifact in software product line engineering [30].

On the architecture level, researches are interested to processes for architecture creation. Reference [28] proposes a design process for modeling and evaluating architectures. In [5], considers variability in the modeling of architecture by the extension of UML models for architecture development. Reference [31] describes patterns in product line architecture design. Reference [32] proposes different architecture views to reduce the complexity of architecture by making the architecture manageable and tailoring the description of architectures to specific stakeholders.

When we look at the implementation level, [33] proposes the transformation of models into text representing source code by transforming different associations for class diagrams into text and generate code. In addition to code generation, aspect-oriented programming and feature oriented analysis can be combined in the implementation of product line asset [34]. Furthermore, generative programming was proposed for the implementation of product lines by allowing the generation of source code for specific products (using compiler flags) [35].

Verification and Validation concerns whole software development lifecycle. Reference [36] discusses the derivation of test cases from use cases containing variability to verify if the implementation corresponds. In [37], the requirements verification for feature models is done by the use of logical expressions representing feature models.

Management level concerns the configuration and versions management by keeping track of versions and traceability. Traceability elicits relations and dependencies existing between

artifacts generated through the software development lifecycle. Some works treat the traceability aspect. In [38], the authors try to record the traceability between solution space and problem space. Reference [39] uses traceability to manage the SPL evolution and to analyze the reasons and the nature of changes in SPL development. In [40], the authors propose a meta-modeling approach to trace variability between requirements and architecture in SPL. Reference [41] study the interaction of MDD and SPL with respect to traceability by the categorization of traceability mechanism.

3 The Framework

The major objective of our research is to acquire and to improve our understanding of the field of software product line. We aim also to identify potential contributions to engineering methods. Our research approach consists on studying the state of art of software product line context and the different concepts related to this topic. We study also the approach of comparison framework of the four worlds which is used in many engineering works in literature [10] [11] and has proven his efficiency in improving the understanding of various engineering disciplines. We used it to propose a comparison framework for software product line.

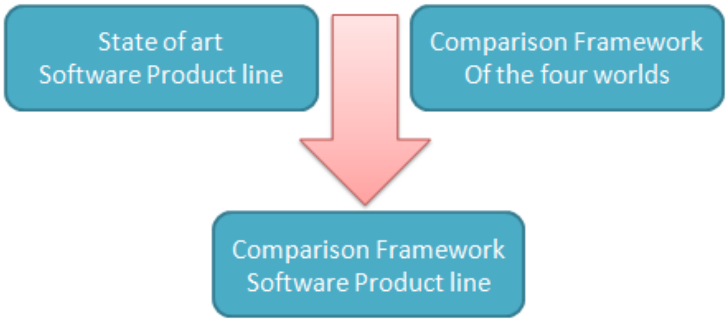

Fig. 4. Research approach. This shows our research approach consisting in the study of state of art of software product line and adopting the approach comparison framework of the four worlds to elaborate our framework for software product line.

We have elaborated a framework to compare different approaches for the construction of SPL. The idea to consider a central concept (here the SPL) on four different points of view is largely inspired from [11], a work dealing with the Web Engineering. Defining a comparison framework has proved its effectiveness in improving the understanding of various engineering disciplines. Therefore, it can be helpful for the better understanding of the field of engineering SPLs. To elaborate our comparison framework, we have proceeded to an analysis of issues that are crucial for the amelioration of the SPL development method. As a result, our framework contains 12 attributes organized into 4 views (cf. Fig. 5) developed in the following subsections.

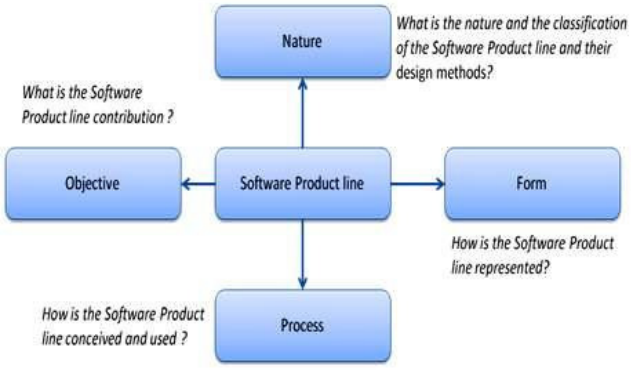

Fig. 5. Software Product Line comparison framework. This shows a figure consisting of our comparison framework with its different views.

Each view allows us to capture a particular aspect of SPLs engineering. Each view is characterized by a set of facets that facilitate understanding and classification of different aspects of SPLs engineering. The facets are defined using attributes which is described by a set of value for measuring and positioning the observed aspect. The facet approach was used in [10] to understand and classify approaches based on scenarios in the field of requirements engineering.

The multi-faceted and multi-view approaches adopted here, allow us to have an overview of the SPLs engineering. The views can show the variety and diversity of the different aspects of SPLs engineering. The facets provide a detailed description of these aspects.

The four views up of the comparison framework respond to four questions about the SPLs engineering:

- What are a method and an application?
- What is the objective assigned to the methods?
- How are represented the construction methods of SPLs?
- How to develop applications?

3.1 Objective View

This view captures why we should use a specific construction methods for SPL and what are the benefits retrieved from its practical application. This view is related to the objectives we seek to achieve in the field of SPL engineering.

3.1.1 Goal facet

A development approach of SPL can be classified in relation to its role. Such approaches were designed for different purposes and try to describe the process in different attitudes: descriptive, prescriptive or both of them. **Prescriptive methods** allow the prescription of how the process should or could be done. **Descriptive methods** allow the study of existing processes and provide explanations about how the development process was conducted. However, certain approaches may include the two strategies as cited in [18]: “Persons dealing with software processes can adopt two different attitudes: descriptive and prescriptive”. This aspect is captured by the **Goal facet**.

As mentioned before, a development approach of software product lines can be classified in relation to its role in Goal facet with the following attribute: **Goal: SET (ENUM {descriptive, prescriptive})**

3.1.2 Policy Management methods facet

Moreover, since the applications change and evolve over the time, it is essential that the methods support these evolutions. The environment of SPL is constantly undergoing to technological, functional (in user needs), structural and behavioral change. Therefore, SPLs are constantly evolving which is important to consider in their development. The evolving systems are scalable systems that can be adapted and extended. This means that a design method for SPL must support evolution. This development should be managed in a controlled flexible and consistent manner. As with any software development, reuse is an important aspect, which can occur at any stage of development and at any level of abstraction and may involve any element of design and/or implementation. There is a growing interest to the identification of reusable design components and design patterns that capture the expertise and experience of designers. Now, the reuse is a part of the policy management methods in the organization. This aspect is captured by the **Policy Management methods facet**.

The elements presented above are captured by two attributes Evolution and reuse of the policy management methods facet:

- **Evolution: Boolean**
- **Reuse: Boolean**

3.2 Form View

The form view deals with different aspects that describe the SPL. This view concerns the methods representation. Indeed, we focus in this view on some points of interest which are:

- What has to be represented?
- How will it be represented?
- At what level of abstraction?

In this view, we found many facets: Models, Notation and Abstraction level.

3.2.1 Models facet

The **Models facet** describes the various models to describe the methods. This facet describes the various models proposed by the method i.e. the different aspects taken into account when designing a SPL. Most of existing design approaches for SPL considers the design phase as an activity that focuses on producing models. The **requirements modeling** for the product line is necessary to try to reach a flexible solution tailored to different user requirements for products of this line to build. The **architecture modeling** of SPL is indispensable for the construction of the product line. The architecture of product lines is an architecture consisting of components, connectors and additional constraints [17]. The role of the architecture of product lines is to describe commonality and variability of the products in the product line and to provide a general common structure. The derived products are the instantiations of the product line architecture and the architecture component. The **user model** is built in order to determine the characteristics of users. It represents the knowledge, objectives and / or individual characteristics such as preferences, interests and needs of users. The **product model** presents the specific features and functionality on which the product must respond.

This facet can be defined as follows: **Models: Set (ENUM {Requirements modeling, Architecture modeling, Product model, User Model})**.

3.2.2 Notation facet

The **Notation facet** concerns how these methods are represented. Indeed, this facet captures the notation nature used in the proposed method (standard, owner or mixed).Some of the studied works are based entirely on standards like UML, MDA…

The Notation Facet helps us to determine the notation used by the method: **Notation: ENUM {standard, proprietary, mixed}**.

3.2.3 Abstraction facet

The **abstraction facet** deals with the abstraction level where these methods are applied. Depending on this level, a SPL model is used as is or will be instantiated. At the level of abstraction of meta-types are all approaches that rely on mechanisms of meta-modeling.

The abstraction facet has one attribute level that takes values in the area listed below: **Abstraction Level: SET (ENUM {type, meta-type})**.

3.3 Nature View

This view answers the « What » question. This means that we will develop facets concerning the internal structure and formalization of the SPL. In this view, we study the nature of SPL and their classification, as well as the definition of methods for their design. In this view, we focus on three facets: SPL's nature, derived products' nature and method's nature.

3.3.1 SPL's nature facet

The first facet presents two types of SPL which are Integration-oriented SPL and open compositional [16]. **Integration-oriented** SPL methods are based on a centrally maintained reusable asset base. This reusable asset forms the basis for any derived products. The reusable assets contain the SPL architecture with a set of components and variation points.

Open compositional SPL methods are based on a distributed software ownership, different goals of parts owners and sub-contracting of part of the software. In this approach, product

developers select partially integrated components and combine it with their own components. However, there is no pre-integrated platform and product developers are free to select the components which suit their needs from the available components.

This facet can take its values in an area listed: **Software product line nature: ENUM {Integration-oriented, Open compositional}**

3.3.2 Products' Nature facet

The **facet products' Nature** can describe the different products resulting from SPL (component, service, constraints on the product, product description, architecture…).

The facet Products' Nature can take its values in an area listed: **Products' Nature: ENUM {component type, service type, restrictions on the product type, product descriptions type, architecture type}**.

3.3.3 Method' nature facet

The proposed research in the field of SPLs engineering are designed to explore different ways to structure the development process, using models and tools tailored to the needs and specificities of the SPL. Some classifications have tried to show the diversity of these approaches in terms of their level of abstraction, granularity and focus. The **method nature facet** deals with these classifications (user-driven approaches, model-oriented approaches, user-centered approaches).The **user-driven approaches** try to identify and define the needs of target users through the step of requirements analysis. Two main techniques are used to determine the user needs: the use cases and the scenario-based approach. **Model-oriented approaches** are model-driven methods. Indeed, modeling the application domain is the starting point of their approach in the construction of SPL. In **user-centered approaches**, the user concept is privileged since the first step in the development process in order to capture the knowledge about the target audience of the future system and to model subsequently the scope of information pertaining to these users. Indeed, a step of user modeling is included in the life cycle approach to identify and describe the different classes of target users to determine the requirements for each category.

The method nature facet is defined as follows: **Method Nature: ENUM {user-centered, model-oriented, user-driven}**.

3.4 Process View

The process view considers different ways of construction methods for SPL conception and usage. Managing variability in SPLs occurs at different levels of abstraction during the development cycle of the product line. All possible variants decrease throughout the development phases. The more we advance in the development, fewer decisions are to be taken towards the possible variation.

A variation belonging to a particular level of abstraction, may give rise to one or more variations located in the lower levels. It must have traceability links among different levels of variation. This helps the identification of variation points to treat after selecting an option belonging to a particular level of abstraction. The choices made at different levels of abstraction can keep the most relevant variants. The number of variants depends from the nature of the system to build. For example, an ERP must be highly variable to suit the needs of several types of businesses.

3.4.1 Lifecycle Coverage facet

The **Lifecycle Coverage facet** deals with the development cycle of a SPL. There is still no consensus on a general model of this development cycle. SPL engineering is defined in the literature [12] by distinguishing two levels of engineering: Domain Engineering and Application Engineering as presented in Fig. 6.

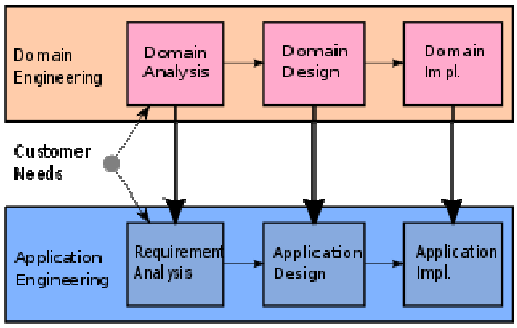

Fig. 6. Software Product line Process. This shows the development lifecycle of SPL.

Domain Engineering (The engineering for Reuse) corresponds to the study of the area of product line, identifying commonalities and variabilities among products, the establishment of a generic software architecture and the implementation of this architecture. Indeed, the domain engineering consists on the development and construction of reusable components known as asset (specification document, model, code ...) which will be reused for the products building. For this reason, the domain engineering is considered as development for reuse. **Application Engineering** (The Engineering by Reuse) is used to find the optimal use for the development of a new product from a product line by reducing costs and development time and improve the quality. At this level, the results of the domain engineering (characteristics model, generic architecture, and components) are used for the derivation of a particular product. This derivation corresponds to the decision-making towards the variation points. It is a development by reuse. The derivation of a particular product requires decisions or choices associated with different variation points detected.

In these levels of engineering, we fund two sub levels: problem domain and solution domain. The process of problem domain analysis consists on the creation of a model describing the problem to be solved. The solution domain defines the environment where the solution is developed and the problem resolution. In domain engineering, these sub levels tries to express structure and variability in software product line (identify variation points and the associated variations and express dependencies between variants to avoid conflicts) and to organize and build reusable artifacts engineering (requirements, tests, components...). In application engineering, variant modeling and solution binding are the next steps. We notice that requirements specifications are important throughout the software product line lifecycle (when defining the entire software product line or a particular product) and the traceability allows the control of specifications compliance.

Software Product Line is considered as effective approach to benefit from software reuse. Configuration management takes more and more a special implication in software product line context as an integral part of any software development activity. The configuration management concerns the two domains, the domain engineering and the application engineering. The challenges for configuration management in software product line consist on configuration artifacts determination, evolution management, and product line problems prevention. Configuration management establishes guidelines for problem tracking (problem domain) and problem resolution process (solution domain). It's considered as transversal activity in lifecycle process which has impacts on domain engineering and application engineering.

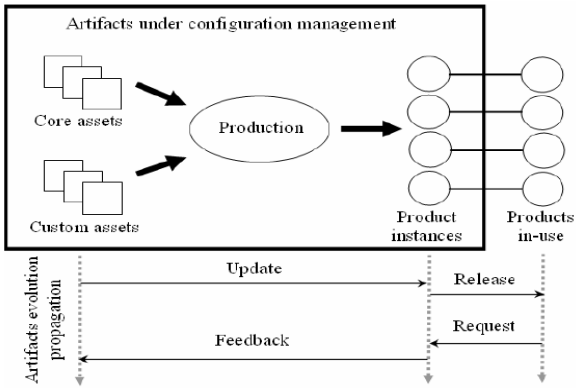

Fig. 7. Configuration management and asset evolution model for software product line [42]. This shows a model of configuration management which makes the changes, maintenance, and evolution controlled.

The Lifecycle Coverage facet can take its values in the area listed below: **Lifecycle Coverage: SET (ENUM {Domain engineering, Application Engineering, Configuration management}).**

3.4.2 Construction technique facet

The **construction technique facet** tries to captures the nature of the used techniques which are based on the instantiation of meta-models, assembly (components, services or COTS), languages and ad-hoc. **Instantiation of a meta-model** is based on the meta-modeling which consists on the identification of common and generic features of products and then represent them by a system of generic concepts. Such representation has the potential to 'generate' all applications sharing the same characteristics. **Assembly technique** consists on the building of a base of reusable components and the assembling of them in products derivation. The reusable components can be either code components or design component. **Language**: The software engineering community has used different languages to develop software applications. This technique has been adopted in the field of SPLs (i.e. compilation directives, template, inheritance…). A sample for the use of compilation directives is shown in Fig. 8. **Ad-Hoc**: Some construction approaches for SPLs are based on expression of the developers' experience. While this experience is not formalized and does not constitute a knowledge base available for the individual developers, we can say that such applications are the result of ad-hoc construction technique.

```
#ifdef VariantA                 /* optional part */
#include CodeForVariantA
#endif
#ifdef VariantB1                /* alternative part */
#include CodeForVariantB1
#else
#ifdef VariantB2
#include CodeForVariantB2
#else
/* default functionality for VariantB goes here */
#endif
#endif
```

Fig. 8. The use of compilation directive. This shows an example of the use of some compilation directive to derive a variant of products.

The construction technique facet can be defined as follows: **Construction Technique: SET (ENUM {Instantiation of meta-model, assembly, language, Ad-Hoc}**).

3.4.3 Runtime support facet

The **Runtime support facet** permits the determination if the approach is supported by a tool. The world of development concerns, in addition to the construction of product line and

derived products, necessary assistance for their implementation and execution. Tool environment is necessary to help the implementation of methodological process and is also a part of development world problems.

This facet is therefore an attribute. This attribute is tool support that allows determining whether the approach is supported by a tool. This attribute takes as value the name of the tool: **Tool support: TEXT**.

3.4.4 Adaptation facet

The **adaptation facet** tries to find the different dimensions that can be adapted in a SPL: features, structures, behaviors and operating resources. Indeed, the heterogeneity of users is observed on several levels: their goals, their knowledge, skills, preferences in terms of features, product structures, behaviors or physical resources from which they access the application. Therefore, it is necessary that the product lines are adapted to their users.

We described the different dimensions that can be adapted in a software product line. Thus, we define the attribute Adaptation as follows: **Adaptation: SET (ENUM {features, structures, behaviors, operating resources})**.

4 Framework Application

Several methods for the construction of SPL have emerged in the literature. Before applying our comparison framework to these methods, we give their brief overview in the following sub-section.

4.1 Overview of Existing Methods

4.1.1 Van Gurp Method

Reference [3] considers the variability as the key to software reuse. The reusable software is distinguished from normal software by supporting various types of variability. Reference [3] has provided a terminology for reasoning about variation points. He described, too, the influence of SPLs approach in the development process. He presented, also, a method to identify, plan and manage the variability in a SPL. This method is based on identification of variation points through the creation of features diagrams. Then, an evaluation phase of variation point's properties is established. The binding of variation points to a variant is the last step of the method which consists in the instantiation of a class and assigns an instance to a property.

According to our framework, this method has as objective a prescriptive attitude to describe process and has the ability to manage the evolution and the reuse by the evaluation phase of variation points. It offers the product model as output and it uses proprietary notation (features diagrams). This user-driven method is based on oriented-integration strategy for the construction of the SPL. In its process, it uses the instantiation, language and Ad-Hoc construction techniques without a tool support.

4.1.2 Ziadi Method

Reference [5] proposes to model SPL variability in static diagrams (use case diagrams and class diagrams) and dynamic diagrams (sequence diagrams using the composition operators of UML 2.0). To ensure the consistency of the SPL model, there are structural rules expressed in OCL, which any SPL must respect. These constraints can be generic or specific to a SPL. Reference [5] proposes an approach for the products derivation (moving from the product line to a particular product) based on the automatic generation of state machines from sequence diagrams. This generation is possible with the use of algebraic specification.

This method supports the evolution and the reuse and offers the ability to construct architecture and product model using a standards notations (UML, OCL). As output, it makes product and architecture description and constraints on the products. This method is a model-oriented method and covers the entire product line lifecycle by using instantiation of model and Ad-Hoc techniques. It allows the adaptation of features, structures and behaviors.

4.1.3 Deelstra Method

Reference [4] proposes the use of model driven architecture (MDA) to manage variability in SPL as presented in Fig. 9. In this proposal, MDA is used as an approach to derive products in a particular type of SPL (configurable product line). The contribution of this study was to combine MDA with a configurable SPL and present the benefits of this relationship. In agreement with the management of variability, two main benefits of applying MDA to the product line engineering are identified, namely the independent evolution of domain concepts, the components of the product line, technology processing and infrastructure used and the use and management of variability as a solution to the problem of round transformations in MDA.

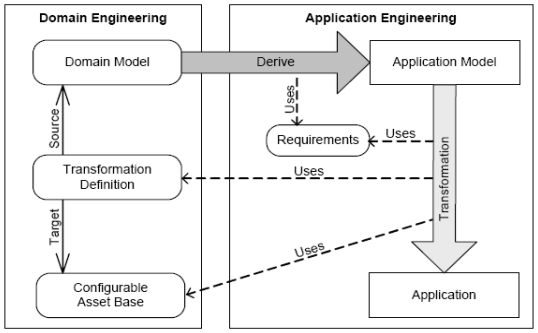

Fig. 9. The use of MDA in Product line engineering. This shows the integration of MDA in SPL lifecycle.

This method supports the evolution and reuse by the use of MDA. It proposes a prescriptive attitude to describe the software process. According to the form view, it produces the architecture and product model with a standard notation (MDA) and in meta-types as level of abstraction. The nature of the derived product of this model-oriented method is component, service, product description and architecture. Its process covers the entire lifecycle and uses instantiation as construction technique without a tool support. Finally, it tries to adapt the operating conditions by the use of MDA which can handle both the platform and technical variability.

4.1.4 Djebbi Method

The RED-PL approach [6] is developed to provide a response to the manner to ensure the satisfaction of the real user's needs and the derivation of the optimal product requirements set (the right product to build). Reference [6] tries, throw this approach, to take into account new stakeholders' needs during requirements derivation for product lines. The RED-PL approach stands for "Requirements Elicitation & Derivation for Product Lines". The matching process tries to do compromise between the product line requirements and the satisfaction of users' needs. The matching process tries to do compromise between the product line requirements and the satisfaction of users' needs.

This method aim for a both prescriptive and descriptive attitude with reference to the process description and it supports the evolution and reuse by the matching process. According to the form view of our framework, it proposes a requirement model besides the product model in a proprietary notation (features diagrams) and a type abstraction level. This method is user-centred because the user concept is privileged. It tries to construct an oriented-integration SPL based on a centrally maintained reusable asset base. It covers only the domain engineering with a Ad-Hoc construction technique and tries to adapt the features and behaviors in agreement with user's needs.

4.2 Comparative Analysis within Framework

The various methods that we have presented approach the design and construction of SPL from different angles. We're going in the first sub-section to summarize characteristics of

different methods according to the four views of our framework to pass in the second sub-section to present a number of shortcomings of these methods.

4.2.1 Evaluation Summary

The table Tab1 presents a comparative analysis of the four selected SPL construction methods. Every method of those presented in sub-section 3.1 covers one particular aspect of the construction of SPL. Reference [3] tries to manage the variability variation point in product line by using features diagrams. Reference [5] is focusing the design aspect by using UML profile (to model SPL and then manage variability in static and dynamic UML diagrams) and OCL constraints (to specify generic or specific constraints on products). Reference [4] proposes the use of MDA to handle both the platform and technical variability by the focus on the design aspect. Reference [6] tries to cover the requirement aspect by the requirement elicitation and the study of constraints in order to ensure the satisfaction of the real user's needs and the derivation of the optimal product.

4.2.2 Drawbacks of Existing Method

The framework analysis allows identifying the following main drawbacks of existing SPL construction methods. We realize that we have a lack of sufficient tool support for them and for their interactivity with their users. The SPL approaches themselves are not enough automated for deriving automatically a product from a SPL. Moreover, some of the proposed methods are using proprietary notations which can handle some problems of standardization and interoperability… In addition, these methods didn't cover all aspects of SPL engineering. Indeed, every method tries to focus on a particular part of SPL construction process. For example, [5] focus his works on the design of SPL and the derivation of products. Reference [6] is working on requirements engineering for SPL to take into consideration the real users needs. In other hand, all these methods offer a prescriptive process that dictates to the designer to perform activities to achieve a particular task. This guides the designer. However, it restricts its participation in the design process. Indeed, such process does not afford him the opportunity to be active in choosing alternatives for achieving its objectives. Finally, in these method, apart [6] ones, the problem is the matching between users' needs and the product offered by developers. The difficulty in mapping lies in the differing languages in which the two parties involved, developers and customers are accustomed to express their self. Many writers have observed that there is a "conceptual mismatch" [13] [14] [15]. Indeed, the developer is placed in operational level, while on the other side customers are placed at the intentional level.

5 Experimentation

Enterprise resource planning (ERP) describe a business management system that integrates the enterprise activities including planning, manufacturing, purchasing, controlling and maintaining inventory, tracking orders... ERP can include application modules for finance and human resources management. ERP systems can improve the performance of the organizations' resource planning, management and operational control. As such, ERP systems are now being developed and evolved for organizations regardless of its size. Return on Investment (ROI) and Return on Values (ROV) depends on the agility of the company to evolve, maintain, and customize/configure its ERP to respond to new business needs and emerging market segments. Evolutionability, maintainability, and configurability are thus at the heart of the ERP business. Software Product Line fits to ERP business. ERP systems can benefit greatly from the concepts of commonalities and variabilities to enhance evolutionability and maintainability. Product-line concepts can reduce complex configuration procedures. We try to adopt product line architectures for the development of ERP systems. We choose to investigate about techniques for modeling, developing, and implementing ERP systems. Studying software product line in a concrete domain as ERP can be useful for the proposal of new method for the construction of software product line.

In this section we briefly present our actual axis of research which is the experimentation. Actually we try to analyze many existing open source ERP in order to find the best solution

with possibilities of evolution. We found two major solutions which are Compiere and Openbravo. Every one of these solutions has many possibilities of customizing and extensions by the adding of new components (reports, helpdesk, task management, payroll…) or customizing the existing ones (Customer Relations Management, Partner Relations Management, Supply Chain Management, Performance Analysis, Web Store…) as presented in Fig. 10 and Fig. 11. Since their inception, Compiere and Openbravo have provided an alternative to expensive and closed ERP systems from Oracle, SAP...

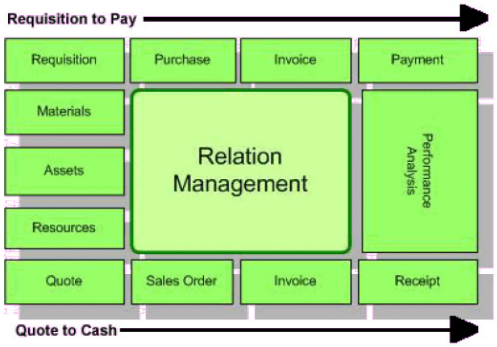

Fig. 10. Domain covered by ERP. This shows management possibilities in an ERP solution.

Compiere and Openbravo are two complete business solutions for small-to-medium enterprises, particularly those in the service and distribution industry, both retail and wholesale. These solutions have an integrated Web Store, covering material management, purchasing sales, accounting, and customer relations management.

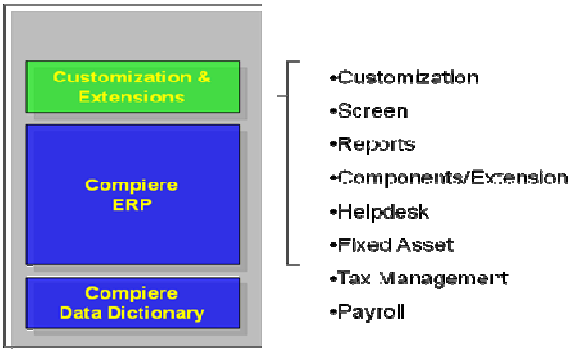

Fig. 11. Compiere specificities. This shows possibilities of customization and extensions in the Compiere ERP solution.

Our methodology consists, as first step, on the evaluation of Compiere and Openbravo. Also, we try to analyze the functional and technical choices done in these solutions to detect the possibilities of evolution according to the real requirements of users by the testing and getting them into production. And finally, we aim to customize and extend the ERP to meet these specific requirements. This final step needs an intentional study to avoid "conceptual mismatch" [13] [14] [15] between developers and customers. Indeed, the developer is placed in operational level, while on the other side customers are placed at the intentional level.

6 Conclusion and future work

In this paper, our contribution was the definition of a comparison framework which has allowed identifying the characteristics and drawbacks of some existing methods for the construction of Software Product Line.

The suggested framework allows a comparison structured in four views. It was build to respond to the following purposes: to have an overview of existing Software Product Line

construction methods, to identify their drawbacks and to analyze the possibility to propose a better method.

Based on this framework analysis, we propose to improve the method used for software product line construction in order to overcome the following drawbacks of existing methods by proposing a tool support to improve interactivity with users. Also, we will try to cover the overall lifecycle of software product line. To avoid the conceptual mismatch, we will try to establish the matching between users' needs and the product offered by developers by the expression of users' needs in an intentional way.

Our future works include a case study of a particular software product line ERP to try to identify variability in this solution. After that, we will try to change in abstraction by studying the ERP in an intentional level to attempt to find the real users' needs and to model the user.

References

1. SEI Product Line Hall of Fame web page, http://www.sei.cmu.edu/productlines/plp_hof.html.
2. D. M. WEISS AND C. T. R. LAI, "Software Product-Line Engineering. A Family-Based Software Development Process". Addison-Wesley (1999).
3. J. Van Grup, "Variability in Software Systems, the key to software reuse". Sweden: University of Groningem (2000).
4. S. Deelstra, M. Sinnema, J. van Gurp, J. Bosch, "Model Driven Architecture as Approach to Manage Variability in Software Product Families", Proceedings of the Workshop on Model Driven Architecture: Foundations and Applications, University of Twente (2003).
5. T. Ziadi, "Manipulation de Lignes de Produits en UML ". Rennes: Université de Rennes (2004).
6. O. Djebbi and C. Salinesi, "Single Product Requirements Derivation in Product Lines" CAiSE (2007).
7. Ø. Haugen, B. Møller-Pedersen, J. Oldevik, A. Solberg "An MDA-based framework for model-driven product derivation", Software Engineering and Applications, MIT Cambridge, USA (2004).
8. M. Sinnema, S. Deelstra, J. Nijhuis and J. Bosch "COVAMOF: A Framework for Modeling Variability in Software Product Families". Proceedings of the Third Software Product Line Conference (SPLC 2004), Springer Verlag Lecture Notes on Computer Science Vol. 3154 (LNCS 3154) 197-213 (2004).
9. J. Lee and K. C. Kang "A Feature-Oriented Approach to Developing Dynamically Reconfigurable Products in Product Line Engineering". Software Product Line Conference (2006).
10. C. Rolland, "A Comprehensive View of Process Engineering", Proceeding of the 10th International Conference CAiSE'98, LNCS 1413, Springer Verlag Pernici, C. Thanos (Eds), Pisa, Italie, p. 1-24 (1998).
11. S. Selmi, "Proposition d'une approche situationnelle de développement d'applications Web". Université de La Manouba : Ecole Nationale des Sciences de l'Informatique (2006).
12. K. Czarnecki and W. Eisenecker, "Generative Programming: Methods, Tools, and Applications". Addison-Wesley (2000).
13. S. N. Woodfield. "The Impedance Mismatch Between Conceptual Models and Implementation Environments", ER'97 Workshop on Behavioral Models and Design Transformations: Issues and Opportunities in Conceptual Modeling, UCLA, Los Angeles, California (1997).
14. Object Matter, www.objectmatter.com.
15. R. Kaabi, "Une Approche Méthodologique pour la Modélisation Intentionnelle des Services et leur Opérationnalisation". Sorbonne: Université de Paris I (2007).
16. J. Van Grup, C. Prehofer, J. Bosch. "Comparing Practices for Reuse in Integration-oriented Software Product Lines and Large Open Source Software Projects". Software: Practice & Experience (Wiley) (2009).
17. L. Bass, P. Clements, R. Kazman, "Software Architecture in Practices", Addison–Wesley (1998).
18. J. Lonchamp, "A structured Conceptual and Terminological Framework for Software Process Engineering", Proceedings of IEEE International Conference on Software Process (1993).
19. M. Jarke and K. Pohl, "Requirements Engineering: An Integrated View of Representation, Process and Domain", in Procedings 4th Euro. Software Conf., Springer Verlag (1993).
20. R. Deneckère, A. Iacovelli, E. Kornyshova and C. Souveyet, "From Method Fragments to Method Services" Exploring Modeling Methods for Systems Analysis and Design (EMMSAD'08), Montpellier: France (2008).
21. S. Ouali, N. Kraiem and H. Ben ghezala "A Comprehensive View of Software Product Line", ICMSC, Egypt (2010).
22. P. Clements and L. Northrop "Software Product Lines: Practices and Patterns." Addison-Wesley

(2002).
23. McGregor, John D.; Northrop, Linda M.; Jarrad, Salah; & Pohl, Klaus. "Guest Editor's Introduction: Initiating Software Product Lines." IEEE Software 19, 4 (July/August 2002): 24-27.
24. David Lorge Parnas. On the design and development of program families. IEEE Transactions on Software Engineering, SE-2(1):1{9, Mar (1976).
25. Jacobson, I.; Griss, M.; & Jonsson, P. Software Reuse: Architecture, Process, and Organization for Business Success. Reading, MA: Addison-Wesley Longman, 1997.
26. Anastasopoulos, M. & Gacek, C. Implementing Product Line Vari-abilities (IESE-Report No. 089.00/E, V1.0). Kaiserslautern, Ger-many: Fraunhofer Institut Experimentelles Software Engineering, 2000.
27. F. Bachmann and P. C. Clements, "Variability in software product lines," Software Engineering Institute, Pittsburgh, USA, Tech. Rep. September, 2005.
28. Thiel, Steffen & Hein, Andreas. "Modeling and Using Product Line Variability in Automotive Systems." IEEE Software 19, 4 (July/August, 2002): 66-72.
29. Krueger, Charles W. "Towards a Taxonomy for Software Product Lines," 323-331. Proceedings of the 5th International Workshop on Product Family Engineering. Siena, Italy. November 4-6, 2003. New York, NY: Springer, 2003. http://www.biglever.com /papers/KruegerTaxonomy_PFE5.pdf.
30. K. Czarnecki, S. Helsen, and U. W. Eisenecker. Formalizing cardinality-based feature models and their specialization. Software Process: Improvement and Practice, 10(1):7– 29, 2005.
31. S. O. Hallsteinsen, T. E. Fægri, and M. Syrstad. Patterns in product family architecture design. In Proc. of the 5th International Workshop on Software Product-Family Engineering (PFE 2003), pages 261–268, 2003.
32. P. America, D. K. Hammer, M. T. Ionita, J. H. Obbink, and E. Rommes. Scenario-based decision making for architectural variability in product families. In Proc. of the Third Int. Conf. on Software Product Lines (SPLC 2004), pages 284–303, 2004.
33. J. Oldevik and O. Haugen. Higher-order transformations for product lines. In Proc. of the 11th Int. Conf. on Software Product Lines (SPLC 2007), pages 243–254, 2007.
34. K. Lee, K. C. Kang, M. Kim, and S. Park. Combining feature-oriented analysis and aspect-oriented programming for product line asset development. In Proc. of the 10th Int. Conf. on Software Product Lines (SPLC 2006), pages 103– 112, 2006.
35. I. McRitchie, T. J. Brown, and I. T. A. Spence. Managing component variability within embedded software product lines via transformational code generation. In Proc. Of the 5th International Workshop on Software Product-family Engineering (PFE 2003), pages 98–110, 2003.
36. T. von der Maßen and H. Lichter. Requiline: A requirements engineering tool for software product lines. In Proc. of the 5th InternationalWorkshop on Software Product-Family Engineering (PFE 2003), pages 168–180, 2003.
37. M. Mannion and J. Camara. Theorem proving for product line model verification. In Proc. of the 5th International Workshop on Software Product-Family Engineering (PFE 2003), pages 211–224, 2003.
38. K. Berg, J. Bishop, and D. Muthig. Tracing software product line variability: from problem to solution space. In Proc. Of the 2005 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries (SAICSIT '05), pages 182–191, , Republic of South Africa, 2005. South African Institute for Computer Scientists and Information Technologists.
39. S. Ajila and B. Ali Kaba. "Using Traceability Mechanisms to Support Software Product Line Evolution". IRI'04, pages 157-162.
40. M. Moon and H. S. Chae, "A Metamodel Approach to Architecture Variability in a Product Line", , 9th International Conference on Software Reuse, Lecture Notes in Computer Science vol. 4039, pages 115-126, Springer Verlag, 2006.
41. Anquetil, N. and Grammel, B. and Galvao Lourenco da Silva, I. and Noppen, J.A.R. and Shakil Khan, S. and Arboleda, H. and Rashid, A. and Garcia, A. (2008)Traceability for Model Driven, Software Product Line Engineering. In: ECMDA Traceability Workshop Proceedings, 12 Jun 2008, Berlin, Germany. pp. 77-86.
42. Softwareproductlines.com, Software Product Lines, http://www.softwareproductlines.com/.

Tab. 1. Appendix: Comparative Analysis of Four Selected Software Product Line construction methods

View	Facet	Attributes	Values domain	Van Gurp method	Ziadi Method	Deelstra Method	Djebbi Method
Objective	Goal	Goal	descriptive, prescriptive	prescriptive	prescriptive	prescriptive	Prescriptive descriptive
	Policy Management methods	Evolution	True, false	True	True	True	True
		Reuse	True, false	True	True	True	True
Form	Model	Model	Requirements Model, Architecture Model, Product Model, User Model	Product Model	Architecture Model, Product Model	Architecture Model, Product Model	Requirements Model, Product Model
	Notation	Notation	standard, proprietary, mixed	proprietary	standard	standard	proprietary
	Abstraction	Abstraction Level	type, meta-type	type	meta-type	meta-type	type
Nature	Software product line's nature	Software product line Nature	oriented-integration, oriented-composition	oriented-integration	oriented-integration	oriented-integration	oriented-integration
	Derived products' nature	Product type	component, service, constraints on the product, product description, architecture	product description	constraints on the product, product description, architecture	component, service, product description, architecture	component, service, product description, architecture
	Method Nature	Method Nature	user-driven, model-oriented, user-centred	user-driven	model-oriented	model-oriented	user-centred
Process	Lifecycle Coverage	Lifecycle Coverage	Domain Engineering, Application Engineering	Not specified	Domain Engineering, Application Engineering	Domain Engineering, Application Engineering	Domain Engineering
	Construction technique	Construction technique	Instantiation, Assembly, Language, Ad-Hoc	Instantiation, Language, Ad-Hoc	Instantiation, Ad-Hoc	Instantiation	Ad-Hoc
	Runtime support	Tool support	TEXT: tool's name	No	No	No	No
	Adaptation	Adaptation	features, structures, behaviors, operating conditions	features	features, structures, behaviors	operating conditions	features, behaviors