QUALITY AND VALUE ANALYSIS OF SOFTWARE PRODUCT LINE ARCHITECTURES

Liliana Dobrica
Faculty of Automation and Computers, University Politehnica of Bucharest
Spl. Independentei 313, Bucharest, Romania

Eila Niemela
VTT Technical Research Centre of Finland, Oulu, Finland

Keywords: Software architecture, analysis methods, quality attributes, product line.

Abstract: The concern of a software product line architecture systematic analysis is how to take better advantage of views and analyze value and quality attributes in an organized and repetitive way. In this approach architecture descriptions evolve from the conceptual level to a more concrete level. Architecture analysis at the conceptual level provides a knowledge base of the domain architecture so as to perform a more comprehensive analysis of quality attributes at the concrete level description. Concrete architecture descriptions permit more relevant and accurate scenario-based analysis results for the development of quality attributes such as portability and adaptability.

1 INTRODUCTION

A product line (PL) is a set of products that together address a particular market segment or fulfil a particular mission. Once a PL is established, new products that belong to the PL can be developed quickly and to a high quality. Although the concept of PL is well known in industrial manufacturing (Tharumarajah et al, 1996), it is a complex and growing research field in software engineering that raises a lot of significant technical and organizational problems. Some of the issues about PL are related to the process of initiation. Software PL does not appear accidentally, but requires a purposeful and definite effort from the organization interested in using a PL approach. The revolutionary initiation into a new PL means that product-line architecture and components are developed to match the requirements of all expected PL members, before developing the first product in a new domain. Product line architecture (PLA) is an adaptable architecture that is applied to a set of products on a PL and from which the software architecture of each product can be derived. PLA includes commonality and variability, indicating what can respectively be common and different among members of a set of PL products. PLA is the first step that shows results of the earliest design decisions about a family of software products. Taking good decisions could lead to reduce costs and risks. In the case of software architecture (SA) for single products analysis methods are mature enough and several have been presented and compared in a survey (Dobrica et al 2002). When considering software PLA analysis methods a strategy for analyzing PLA is introduced in (Dobrica et al 2000). The open problem of a PLA analysis method is how to take better advantage of architectural concepts and analyze quality attributes in software PL in a systematic way. It is also very important to identify potential risks and to verify that the quality requirements of the PL domain have been addressed in the PLA design. The PLA must not only conform to the quality requirements for each PL member, but it must also be generic and adaptable to the whole PL domain. It is important to know how reusable and flexible to anticipated changes PLA is so as to maximize reusability and to minimize possible changes in functionality required by various product members.

Dobrica L. and Niemela E. (2008).
QUALITY AND VALUE ANALYSIS OF SOFTWARE PRODUCT LINE ARCHITECTURES.
In *Proceedings of the Third International Conference on Software and Data Technologies - SE/GSDCA/MUSE*, pages 64-71
DOI: 10.5220/0001888700640071

2 BACKGROUND

2.1 PLA Representation

There are several architectural development approaches that can be adopted in PLA representation. The Model-driven Architecture (Miller J and Mukerji J, 2003) is an approach that guides the specification of information systems. The idea is to separate descriptions of functionality from the implementation specifications. Implementation independent descriptions of functionality last longer than implementation specifications that change as soon as a better technology is available. In MDA, a model means a formal specification of part of the function, structure and/or behavior of a system. A formal specification expects either textual or graphical language with strictly defined syntax and semantics. Other design approaches concentrates on multiple views of an architecture. An architectural view is a representation of a whole system from a perspective of a related set of concerns (ISO/IEC 42010, 2007). View-oriented design approaches start with 4+1 approach (Krutchen, 1995), after which other news have been introduced (Jaaksi A et al, 1999) (Hofmeister et al, 2000). Among these approaches there is no agreement on a common set of views or on the way to describe SA. The need for different architectural views depends on three issues: the size, the domain and the number of different stakeholders. Although a multiple view approach helps in developing software products, it is easy to introduce errors and inconsistencies in a multiple view model. It is therefore necessary to provide support for consistency checking among the multiple views.

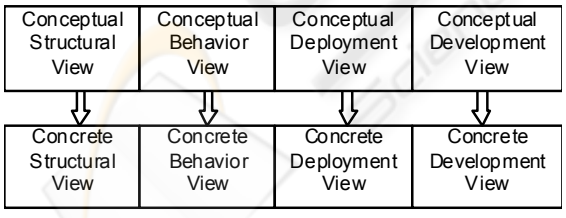

Figure 1: Product line architecture views.

The goal of the design of a PLA is to address every stakeholder's concerns and to satisfy the win conditions of all of the stakeholders. Stakeholders determine the necessary views of architecture. So, the most significant challenge for PLA representations is to support useful and consistent views of the architecture from multiple perspectives. Works has been done on software architectural views and the way to relate to them (Krutchen 1995), (IEEE 2000), and (Purhonen et al 2004). In our method architecture views evolve from the conceptual level description to a more concrete level during design (Figure 1). Conceptual means abstract, i.e. delayed design decisions concerning, e.g. technologies to be selected or details in functionality, whereas the concrete abstraction level illustrates the realization of conceptual architecture. Architecture design produces descriptions at both abstraction levels from four viewpoints: structural, behaviour, deployment and development. The structural view is concerned with the composition of software components, whereas the behaviour view takes the dynamics into consideration. The deployment view refers to the allocation of software components to various computing environments. Variation in space is an integral part of the first three views, contrary to the development view that represents the categorization and management of domains, technologies and work allocation.

2.2 PLA Analysis

In (IEEE 1061,1998) software quality is defined as a degree of software to process a desired combination of quality attributes. The software quality model (ISO/IEC 9126) defines six categories of characteristics (functionality, reliability, usability, efficiency, maintainability, and portability) that are divided into subcharacteristics, which are externally or internally observable properties of systems.

Scenario-based assessment is appropriate for qualities related to software development, which are specific to PLAs. Software qualities such as, adaptability and portability can be expressed very naturally through change scenarios. *Portability is the ease with which a system can be adapted to changes in the technical environment and adaptability is the ease with which a system can be adapted to changes in the technical requirements.* At first sight, portability and adaptability very much look alike, but they are not the same. The use of scenarios for evaluating architectures is recommended as one of the best industrial practices. By formulating a number of scenarios, we can make each quality attribute tangible, because a scenario capture what we actually want to achieve with that quality attribute. However, the evaluation depends on the objectivity and creativity of the analyst who defines and executes them.

The systematic quality analysis for both conceptual and concrete architecture descriptions takes into account scenarios The analysis may

involve multiple views. Utilizing the architectural constructs mentioned above, the method provides an explicit and quality-driven link between software requirements and architecture. The functionality that the products need to support is decomposed in a structural view. At a conceptual level, this view is useful for understanding the interactions between entities in the problem space, planning functionality and understanding the domain *variability*, and hence thereafter, the possibilities of initiating a PL. At a concrete level the elements from which the system is built could be essential for understanding the maintainability, modifiability, reusability and portability of a system. Behaviour view is important to understand not only *performance* but also *reliability* and *security*. Deployment view consists of central processing units, memory, buses, networks or input/output devices. Quality attributes relevant to this view are *availability*, *capacity* and *bandwidth*. Utilizing the architectural constructs mentioned above, the method provides an explicit and quality-driven link between requirements and architecture.

3 PLA SYSTEMATIC ANALYSIS

Before starting the development of a PL, a company has to consider various issues in order to gain an understanding of whether a PL is appropriate for different technologies and businesses (Niemela et al 2001). In order to help make decisions about PL scope, an evaluation that considers an appropriate value metric is needed. The analysis is driven by scenarios, but it identifies which changes are most valuable reported to a market and it quantifies the expected return on making that change. Since it is developing a common architecture for a family of products, the goal is to design an architecture that encompasses all the PL members' common features, but which can be easily adapted to produce any member of the family. This means that addressing the variations among members should require no change, or very little change to the common architecture.

3.1 Value Analysis

The idea is to provide a common framework and metric for making decisions which bring together business issues and product issues. To do so, a value metric that makes sense in all these assumptions is required, to help make decisions about PL scope. The focus could be on including or excluding capabilities from PL scope and measuring a relative benefit, adopting PL solutions based on cost/benefit decisions or considering technologies applicable only due to PL (e.g. realizing a specific software tool needed for a PL). An important element in PL domain definition is the market, represented by customers as stakeholders. Value analysis is similar to quality function deployment (QFD) (Hauser et al 1988) in searching for to harmonize market (i.e. customer) needs with product design. It differs in that it seeks to measure the customer's perception of total delivered value more directly and accurately, i.e. what the customer will actually consider important for the product. Second, it directly measures the difference between an organization's internal understanding of customer value and the customer's actual recognition. This provides a basis for aligning a company internal view of delivered value with market realities. Activities related to value analysis are clustered in domain definition and commonalty analysis (Figure 2).

Domain definition consists of scope, economic analysis and value analysis (VA). Scope considers the creation of a preliminary definition of the PL in terms of commonalties and variabilities. Economic analysis is concerned with the building of an economic model of the product's cost/return using the company's current software products and then PL; these models may be used to determine the expected return from adopting a PL approach. VA is performed to help establish the relative value of the possible variations in the potential scope of the PL. In this context, change scenarios are created based on how the product is expected to evolve to meet market (i.e. customer) needs. The results are used both to identify which changes are most valuable and to predict the expected return on making the change. The change scenarios are used against the current architecture to determine the expected cost of evolving a product without using a PL. Then VA evaluates the costs and benefits of each approach based on the value of the product changes the market wants and the costs of making such changes under each development paradigm.

Commonality analysis. This activity has the goal of identifying and documenting the commonalties and variabilities characterizing the software PL. Also, the VA is refined by developing value metrics based on the more detailed definition of expected variations. This new iteration on VA aligns market value data with the PL requirements. The architecture quality analysis (AQA) is correlated with the design activities that have the result PLA model (Figure 2).

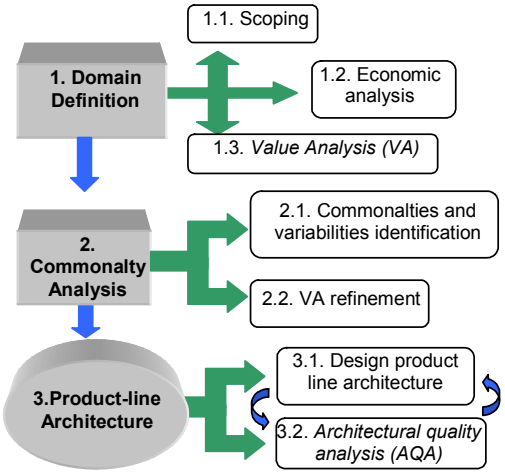

Figure 2: Using value in architectural analysis.

Here, a compositional approach to domain architecture is considered, in which a common, reusable architecture for the PL is developed. In order to generate various members of the PL, adaptable, parameterized components can be used (e.g. as in CelsiusTech architectural case study (Bass et al 1995)). As part of the modeling activity, the architecture must be evaluated against its quality requirements. In particular, a detailed, quantitative evaluation of how well the architecture instantiates the commonalties and accommodates the variabilities that characterize the PL is desirable. The purpose is both to assess the quality of a conceptual or concrete architectural design relative to the requirements and to quantify that measure of quality so it can meaningfully compare different designs. This considers the results of domain definition and commonalty analysis (Figure 2).

3.2 PLA Quality Analysis at the Conceptual Level

This phase focuses on becoming aware of the available and required information to carry out the analysis, and then to collect and assemble it. The PL requirements define not only the PL scope, but also represent the input used to create a knowledge base of requirement taxonomy (Figure 3). Syntactic architectural notations should be well understood by the parties involved in the analysis. The result of an evaluation process depends on how well the description is made. This phase focuses on specific SA analysis and the generation of artifacts to make the analysis. Examples of artifacts include: domain models (which help in comparing competing architectures within the same functional area); relevant architectural views; architectural styles; environmental assumptions and constraints; and trade-off rationale.

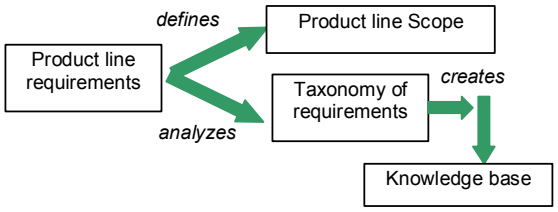

Figure 3: Conceptual software PLA analysis.

The role of a knowledge base is to allow collections of architecture styles and patterns to be evaluated in terms of both quality factors and concerns, and anticipations of their use (Niemela et al 2005). A ”pre-scored” of architectural patterns is feasible in order to get a sense of their relative suitability to meet particular quality requirements of a system. In addition to evaluating individual patterns, it is necessary to evaluate compositions of patterns that might be used in architecture. Identifying patterns that do not compose well (the result is difficult to analyze, or the quality factors of the result are in conflict with each other) should steer a designer away from “difficult” architectures towards those made of well-behaving compositions of patterns. The knowledge base built in this way helps to move from the notion of architectural styles toward the ability to reason (whether quantitatively or qualitatively) based on quality attribute-specific models. The purpose of having a knowledge base is to make architectural design more routine-like and more predictable, to have a standard set of attribute-based analysis questions, and to tighten the link between design and analysis by means that can be used to provide context-dependent measures. After this phase, the activities of the concrete architecture design and the second phase of architecture analysis are performed.

3.3 PLA Quality Analysis at the Concrete Level

In this phase recommendations are made, “hot spots” in the architecture (areas of high predicted complexity, large numbers of changes, performance bottlenecks, etc.) are located and strategies for their mitigation are enumerated, and common reference models are identified. A detailed and quantitative analysis, AQA, is developed by creating scenarios based on the results of commonality analysis and evaluating them based on the results of VA and an analysis of the cost and benefits associated with potential variations in the scope of the family.

The knowledge base is attached to the PL scope in a form of requirements' taxonomy. This is used with the aim to establish how adaptable the PLA is to the expected changes related to this taxonomy. The analysis considers PL specific techniques such as commonality analysis, which systematically models the required similarities and differences among PL members. It is also considered that PLA contains the common components of the architectures of the product members and takes variabilities as possible changes to this. The main inputs of the method are the PL scope and PLA.

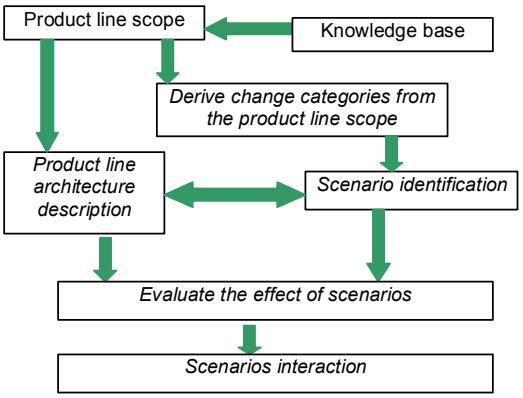

Figure 4: Inputs and activities of AQA.

The method consists of five important steps (Figure 4). The fist step is to derive change categories based on PL scope (Figure 5). A category could contain scenarios that are related to the technical requirements. In this case scenarios explore the applicability of the PLA in situations with various technical requirements, so they represent PLA adaptability. Another scenarios category may concentrate on context identification and may simulate changes in the technical environment, so they represent PLA portability.

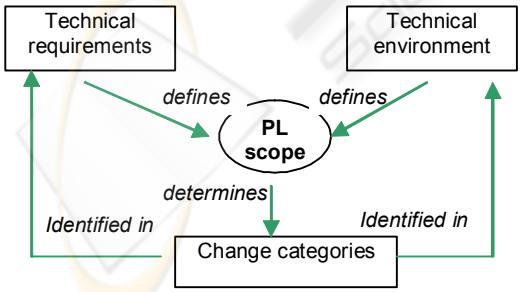

Figure 5: Derived change categories.

Then, the description of the PLA and the scenarios identification are performed in parallel. The simultaneity of these steps enables a decision to be made as to what view should be considered for an elicited scenario. Scenario effects evaluation and interaction are the last steps performed sequentially. An overall evaluation of the architecture may be performed using customer value data to assign weights to scenarios and scenario interactions. This weighting can be used to evaluate one candidate architectural design against another. When we evaluate the effect of scenarios on the architecture, we classify the effect of a scenario into four discrete levels. At the first level, no changes are necessary, which means that the scenario is already supported by the architecture. At the second level, just one component of the architecture needs to be changed. At this level, we have true locality of change. At the third level more than one component is affected, but no new components are added or existing ones are deleted. This means that the structure of the architecture remains intact. At the fourth level, architectural changes are inevitable, because new components are necessary or existing ones become obsolete. It is clear that one should seek to keep the level of effect as low as possible.

4 CASE STUDY

4.1 PL Scope and PLA Description

Distributed services operate in different units that are executed in devices and are organized to operate in the form of a network. The units operate in a collaborative way in order to provide the platform system services. The system services are further utilized through certain interfaces by application servers and users. The purpose of the system services of the platform is to enable application services to distribute themselves smoothly and comfortably. Figure 6 describes DiSeP context diagram illustrating external actors that interface with our platform. The new external actors, *TransactionManager* and *TransactionParticipant* are not mandatory. The context of the DiSeP PLA is as important as the other PLA views because it reveals other new, external actors that can interact with a new potential PL member (Figure 4).

Distributed parts of the application services may locate and utilize each other in a dynamic manner reaching the following technical properties: 1) platform implementation independence, 2) distribution transparency and 3) mobility of system services. The first technical property means independence of implementation languages and a universal communication manner between units. The second property refers to the ability to resist

dynamic changes in configuration of network of interconnected units or in the physical communication links between different devices. The third considers that system services are not centralized into one location, but any one of the units can act as a system service provider. DiSeP is the first model of PL in the domain.

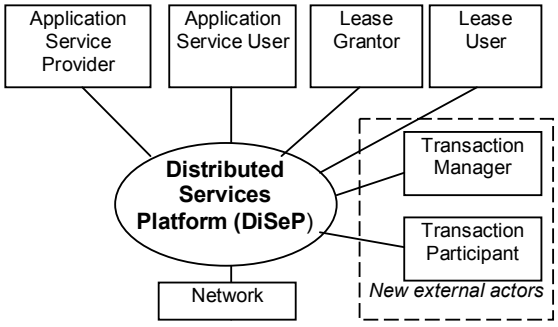

Figure 6: DiSeP Context.

PLA has been documented around multiple views describing conceptual and concrete levels, for each view a static and dynamic perspective being offered. The views were illustrated with diagrams expressed in a real-time extension of UML.

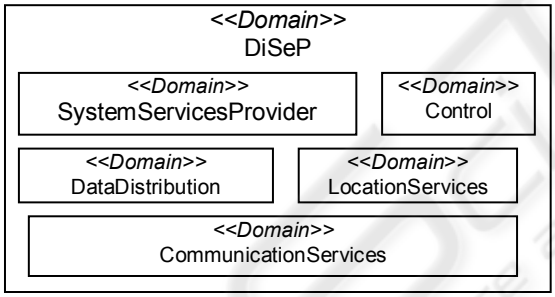

Figure 7: DiSeP domain conceptual view.

The conceptual level considered a functional decomposition of the architecture into domains. The relationships between architectural elements are based on pass control and pass data or uses. The concrete level considered a more detailed functional description, where the main architectural elements are packages, capsules, ports, protocols. The relationships are association, specialization, generalization, etc. The dynamic aspect includes statecharts and message-sequence charts. Layer architecture style is considered for the conceptual structural view. PLA concrete structural view includes abstract components (Figure 8). All the architectural elements are subsystems <<service>> that are common to all product members. The provided services, contained in the *SystemServiceProvider* subsystem, are activated by a *Control* subsystem. These services communicate with the other subsystems, such as *DataDistribution*, *LocationServices* and *CommunicationServices*.

Two variability points are identified in DiSeP PLA, but others are implicit or unspecified. The services inside the *SystemServicesProvider* represent one of the variable points. In some of the products there are two services: LeaseService and DirectoryService that always come together. In other products there also might be SecurityService, TransactionService, etc. A second point of variability is inside the *CommunicationServices* domain. The communication could be performed using *SynchMessService* or *AsynchMessService*.

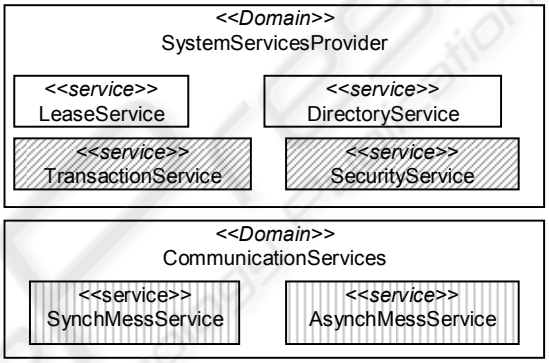

Figure 8: Variability in service components.

4.2 DiSeP PLA Quality Analysis

To assess the quality of the DiSeP architecture, we only use the software architecture analysis (AQA). Due to the revolutionary initiation approach, we are not able to perform VA (that can be done when the first version of the new concept is ready).

As stated above, this is a scenario-based method that consists of formulating a number of scenarios and evaluating the effect of each of them on the architecture. AQA is used to assess the quality at architectural level, namely PLA, that represents the commonalties and encapsulates the variabilites of PL members. The first step in the evaluation is to derive a number of scenarios from the requirements of the architecture included in the PL scope. For example, from the quality requirement portability we can derive the following scenario: *What happens when another network protocol is to be used?* By formulating this scenario, we can make portability tangible, because it captures what we actually want to achieve with portability. The next step is to evaluate the effect of these scenarios on the architecture as described in the previous section. We see that our example scenario demands a change in

the *CommunicationService* component. Thus, this scenario has a level two effect. It means that we have locality of change for this scenario and that the architecture is portable with respect to the network protocol used.

We have created two categories of change associated to adaptability and portability. Portability is a quality requirement placed in the technical environment category and adaptability represents the flexibility of DiSeP to incorporate changes to its technical requirements.

Adaptability. The scenarios simulate the use of the DiSeP architecture in situations with diverse technical requirements. The architecture is usable in a situation when the scenario has an impact of level three or lower.

Scenario 1. Which changes are needed when the architecture is to be used in secure systems?

We assume that for secure systems a number of things are necessary. First, each service user/provider/grantor action should be authenticated and it should be possible to grant different levels of access to users (no access, read-only, full control, etc.). This is already supported by the distributed service platform architecture, so it is unaffected. Second, the communication between components should be encrypted. Encrypted communication is not yet present in the architecture, but it could be added by changing one component – communication service. Finally, access to services should be prohibited for unsecured units. This means that the location service manager should be changed so that it inspects the network addresses of clients. The conclusion is that using the architecture for secure systems demands changes to a number of existing components and, therefore, this scenario has a level three impact. Other scenarios could be:

2. Which changes are needed when the architecture is to be used in real-time systems?

3. Which changes are needed when the architecture is to be used in ultra-reliable systems?

4. Which changes are needed when another type of interface is considered for a service user?

5. Which changes are needed when the architecture is used in a system that uses workflow management?

6. Which changes are needed in a system that uses mobile computing?

The results are summarized in Table 1. As expected, we see that the architecture is not directly usable in every situation. Using it for real-time or ultra-reliable systems requires major changes to the architecture. In the other situations, the architecture is usable, but some changes are needed. When the DiSeP architecture is used in an actual situation, more scenarios are probably required to evaluate whether the right services are identified in order to encapsulate the expected changes to the technical requirements. Initiating the PL is a highly iterative process. This analysis leads us to the conclusion that first, a concrete functionality should be designed and then attention to this quality attribute should be paid.

Table 1: Summary of the scenarios adaptability. (- = unaffected, + = needs to be changed, O = one comp affected, M = more comp affected).

Scenario	DiSeP		Impact level
	Architecture	Components	
1	-	M	3
2	+	M	4
3	+	M	3
4	-	M	3
5	-	O	2
6	+	M	4

Portability. We exemplify with scenarios that explore the effect of changes in the technical environment.:

1. Which changes are needed when another end point device is used?

2. Which changes are needed when another network protocol is used?

Table 2: Summary of the scenarios for portability(- = unaffected, + = needs to be changed, O = one component affected).

Scenario	DiSeP		Impact level
	Architecture	Components	
1	-	-	1
2	-	O	2

In Table 2, we observe that changes in the technical environment affect very few of the DiSeP architecture components. We notice that the platform actually encapsulates access to the environment. However, there may be potential changes in the technical environment, not mentioned here, that have an impact above level two.

5 CONCLUSIONS

In this paper we have introduced an analysis method of a PLA that has been described in multiple views on two abstraction levels. Our main and original contribution is that we consider both economic and quality aspects in this systematic analysis. The work has been motivated by increasing realization in the software engineering community of the importance

of PL from economic viewpoint and SA for fulfilling quality requirements. Quality analysis at the conceptual level examines the relationship between architectural views and architectural styles, as an architectural style is also considered to have an impact on quality attributes of the system. In this way, the result of the examination responds to questions such as: (1) upon what architectural view does the architectural style focus, (2) what specific quality attributes the style is considered to support, and (3) what kind of assumptions are made about context or environment. Assuming that there are already known benefits and drawbacks of each style in relation to quality attributes, the analysis of conceptual descriptions has the aim of checking styles and violations to the standard patterns. Also, the role of analysis at the conceptual level is to provide a knowledge base of the PLA so as to perform a more comprehensive analysis at the concrete level description. Thus, the experts' knowledge could be better structured and used in a more systematic way to generate scenarios associated with the most important quality attribute of the domain. Towards an architectural knowledge base for wireless service engineering some progress has been made and described in (Niemela et al, 2005). The quality analysis of the concrete architecture makes it possible to obtain better results that improve the design. Concrete architecture permits more relevant and accurate results.

The PLA of the DiSeP is the first stage of a software development cycle and we have tried to model it by means of applying an approach for PL initiation from PL requirements. The development of the DiSeP PLA is an iterative process, so the analysis is as well. One of the goals of analyzing the conceptual design of PLA is its relevancy for uncovered PL features. On the concrete architecture we analyzed adaptability and portability as development quality attributes, using a scenario-based method. We mention that is very important to consider the economic aspects of the analysis. We could not exemplify the value analysis, due to the lack of an economic data model. However a value metric of an economic model is required to make decisions about PL scope and in commonality analysis, too. (Clements, 2007) described SA decisions based on an economic model.

In future research we want to validate this systematic approach in various software application domains where a product line is initiated. More work is needed to develop systematic ways of bridging other quality and economic requirements to a PLA. However this paper presented the main concepts and justified why this concepts are required.

ACKNOWLEDGEMENTS

We wish to thank the anonymous referees for their valuable suggestions and comments.

REFERENCES

Bass L., P. Clement and R. Kazman, (2003) *Software Architecture in Practice*, Addison Wesley, Reading.

Dobrica L., E. Niemelä, (2002) A Survey on Software Architecture Analysis Methods, *IEEE Trans Software. Eng*., Vol.28 (7), 638-653.

Dobrica L., E. Niemelä, (2000), *A Strategy for Analyzing Product Line Software Architecture*, VTT Publications 427, Espoo, Finland, 124 p.

Hauser J.R. and Don Clausing (1988), The House of Quality, *Harvard Business Review*, May-June.

IEEE Std 1471-2000 (2000), IEEE Recommended Practice for Architectural Description of Software-Intensive Systems.

ISO/IEC WD1 42010 (2007), Systems and Software Engineering – Architectural Description.

IEEE 1061 (1998), IEEE Standard. for Software. Quality Metrics Methodology, IEEE Std 1061-1998.

Krutchen P. B., (1995), The 4+1 View Model of Architecture, *IEEE Software*, Nov.,pp. 42-50.

Matinlassi, M., E. Niemelä and L. Dobrica, (2002), *Quality-driven architecture design and quality analysis method - A revolutionary initiation approach to product line architecture*, VTT Publications 456, Espoo, Finland, 139p.

Niemelä E. and T. Ihme, (2001), Product Line Software Engineering of Embedded Systems, *Procs of SSR'01, Symposium on Software Reusability*, pp. 118 – 125.

Niemelä E., Kalaoja J., P. Lago, (2005) Toward an Architectural Knowledge Base for Wireless Service Engineering*, IEEE Trans. Software Eng.*, Vol 31 (5), p. 361 – 379.

Purhonen A. , E. Niemelä , M. Matinlassi, (2004) Viewpoints of DSP software and service architectures, *Journal of Systems and Software*, Vol.69(1-2), p.57-73.

Tharumarajah A, A.J. Wells, L. Nemes, (1996) A Comparison of the bionic, fractal and holonic manufacturing concepts, *International Journal of Computer Integrated Manufacturing* 9 (3) 217-226.

Clements P. (2007), An economic model for software architecture decisions, *Procs. ICSEW'07 International Conference on Software Engineering Workshops.*

Miller J and Mukerji J, 2003, *MDA Guide Version 1.0.1*., Object Management Group, 2003.

Jaaksi A et al, 1999, Tried & True Object Development: Industry-Proven Approaches with UML. Cambridge Univ. Press, 1999.

Hofmeister C et al., 2000, Applied Software Architecture. Addison-Wesley, 2000.