Quality aware Software Product Line Engineering

Leire Etxeberria, Goiuria Sagardui and Lorea Belategi

Computer Science Department University of Mondragon
Informatika departamentua, Goi Eskola Politeknikoa, Mondragon Unibertsitatea, Loramendi 4,
Apartado 23, 20500 Mondragón (Gipuzkoa), Spain
letxeberria@eps.mondragon.edu

Abstract

Meeting and managing quality requirements such as performance, security… in a reuse context (software product line…) has a problematic that it is not found in single-systems. In this paper, an overview of aspects to consider is presented, including a review of existing approaches, as well as some conclusions, requirements and guidelines to address quality aspects in software product lines.

***Keywords**:* Software product lines, quality assurance

1. INTRODUCTION

"Software product line (SPL) is a set of software-intensive systems that share a common, managed set of features to satisfy 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"[13].

In software product line development, two phases are distinguished: Domain engineering and Application engineering. Domain engineering is in charge of developing a common infrastructure and assets while Application engineering makes use of them to generate the products of the line.

While working on domain engineering phase, all the requirements of the products must be taken into account, including quality attribute requirements such as performance, reliability, usability, etc. However, "research in the field of software product lines has primarily focused on analysis, design, and implementation to date and only very few results address the quality assurance problems and challenges that arise in a reuse context" [32]. In a software product line, quality attribute requirements have also variability, because not all the products require the same level of security, performance, etc. This aspect has also been neglected or ignored by most of the researchers as attention has been mainly put in the variability to ensure that it is possible to get all the functionality of the products.

Fortunately, things have started changing and the interest on quality in reuse contexts has grown considerably in the last years; several workshops have been organized: eWorkshop on Quality Assurance for Software Product Lines: Strategic Issues [33], International Workshop on Quality Assurance in Reuse Contexts (QUARC'04) [31], The First IEEE International Workshop on Quality Oriented Reuse of Software (QUORS'07)… and research paper's production has proliferated.

If in single-systems achieving quality attributes is sometimes a challenge, in software product lines this challenge is complicated because there is variability on quality attribute requirements and different quality constraints are required. For instance, variable quality constraints are required due to variability on hardware: the storage space, execution capability, screen's dimensions… of the hardware device can impose different constraints. Tradeoff analysis of quality attributes is also more difficult than in single-systems due to this variability and the exponential number of possibilities.

The difficulty grows but the impact of not addressing quality attributes also does. The consequences of not considering and managing variability in quality attributes when designing a product line are not trivial, especially in embedded systems. If a product line is developed without considering the quality attribute requirements' variability, this product line will not cover all the products of the scope and will probably not cover new products in the future. As a consequence, the investment for developing the software product line will not be cost-effective.

As conclusion, to develop a product line that addresses the customers' needs, quality attributes and their variability must be gathered and managed during domain engineering. This paper presents an overview of the aspects that need to be considered during domain engineering to address quality as well as reviews of

methods and approaches for each aspect. The rest of the paper is organized as explained here. In Section 2, the tasks to meet quality requirements in a software product line are presented. In Section 3, an overview of existing methods and approaches for those tasks are reviewed and in section 4 conclusions and requirements for quality driven domain engineering are drawn. Finally, section 5 presents related work and section 6 future work.

2. ACHIEVING QUALITY IN SOFTWARE PRODUCT LINES

As we have mentioned, it is vital to take into account quality requirements when developing software product lines. First of all, in the next subsection, what means quality in a software product line is explained and the specific classification of quality attributes for a software product line is presented.

2.1. SOFTWARE QUALITY IN SOFTWARE PRODUCT LINES

Software **quality** is the degree to which software possesses a desired combination of attributes [25]. There are two broad categories of quality attributes [4]: **Observable via execution** or **operational** such as performance, security, availability, usability… and **not observable via execution** or **development attributes** such as modifiability, portability, reusability, integrability, testability…

In a software product line, quality attribute requirements can be classified in two different types [18]: *Product-line quality attributes* and *domain-relevant attributes*. *Product-line quality attributes* are considered development attributes or non observable via execution. Whereas *domain-relevant quality attributes* usually are operational or observable via execution.

Product-line quality attributes are those that are inherent or specific to product-lines to undertake a set of related products as well as new future products. These attributes are the ones related to variability or flexibility. Assessing the variability of a product line ensures that it is possible to get all the functionality of the products in the envisioned scope; variability [55] understood as modifiability (to allow variation or evolution over time) and configurability (variability in the product space) to get a set of related products.

Domain-relevant quality attributes (such as safety in a safety-critical domain, performance in a real-time domain, etc.) must also be addressed in the product line, otherwise the implications or consequences can be very serious and difficult to fix. As different products of the domain can require different attribute values (not all products require the same level of security…), variability in the way the attribute is translated to the product is relevant for the assessment to assure that the realization of all the quality attributes for all the products in the product-line scope is possible.

2.2. QUALITY AWARE DOMAIN ENGINEERING

Software product line practice seeks to achieve a number of goals including reduced costs, improved time to market, and improved quality of the products belonging to the product line. These goals will only be achieved if quality attributes, such as correctness and reliability, are continuous objectives from the earliest phases of development [39].

The importance that quality and quality assurance is acquiring in reuse contexts is justified because an error in a reusable asset can be propagated to a lot of products.

Regarding the previous classification of quality requirements, to date, *product-line quality attributes* such as extensibility, modifiability, etc. has received most of the attention: how to know if the line covers all the envisioned functionality of the products in the scope. Nevertheless, *domain-relevant quality attributes* have being neglected, especially the variability that those attributes can have in a software product line. This paper deals with what can be done to address *domain-relevant quality attributes* and their variability when developing a software product line.

To obtain the required quality levels, quality must be addressed from the beginning, not only at design time, which is the stage where quality is usually considered, it must be considered through all the life cycle. During requirement engineering quality requirements must be captured and modelled including variability aspects and during design, these requirements must be taken into account to get the most adequate design. Design's evaluation is also very important because it allows early problem detection in the life cycle. We have made a list of tasks or practices to perform during domain engineering to facilitate quality aware product line engineering (see Figure 1). Some of them are specific tasks and others consist on including a quality perspective in tasks that are performed in all software product line developments.

1. Quality variability modelling: To specify quality variability. Different products are going to have different quality levels and this information should be explicit.
2. Quality aware design: To take into account quality attributes including variability during software architecture design.
3. Architecture evaluation: To evaluate the design (taking into account the variability) to see if all the quality levels are fulfilled.
4. Quality aware implementation: To take into account quality attributes during software coding.
5. Quality testing: To assure that quality requirements are achieved once the product line assets have been developed.

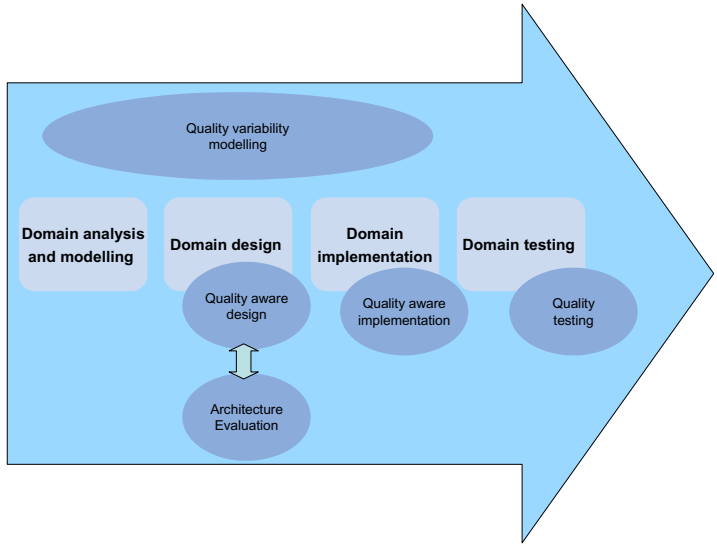

Figure 1: Quality aware domain engineering

Quality variability modelling

In a software product line, different members of the line may require different levels of a quality attribute or a quality can be optional for some products (if the quality is not important for all products). Moreover, functionality and qualities are closely related and the selection of a functional feature can influence or impact on the quality level. Three different aspects of quality attribute variability must be considered, following the classification of quality attribute variability types of *Niemelä* [42]:

- *Optionality of a quality attribute*. For example, for one product of the line, performance is important but for others products of the same line there are not performance requirements.
- *Quality attribute levels or groups*. In a product line, quality attributes can have different priority levels. For example, for one product the performance requirements are extremely high, whereas for others those requirements are at the lowest level.
- *Impacts of functional variability on quality (Indirect variation)*. Functional variability can indirectly cause variation in the quality requirements.

To be able to assure quality aspects, the quality variability must be managed during all the life cycle and for it, quality variability modelling is essential including variability aspects of requirements, design and implementation.

Quality variability model is useful during domain engineering for addressing quality aspects taking into account their variability and also during application engineering when products are derived for facilitating a quality driven derivation.

There are some approaches for specifying quality attribute requirements that do not address variability explicitly. In a similar fashion, there are other approaches to specify varying requirements that do not address quality attributes [40]. However, there are also several methods that address quality attribute variability explicitly.

Quality aware design

During design, the software architecture of the product line is defined. “The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them” [4]. The software architecture has a great influence on the system’s final quality as it can inhibit or enable the product’s quality attributes. For that reason quality requirements and their variability must be very present during architecture design.

Architecture evaluation

The architecture evaluation is “the systematic examination of the extent to which an architecture fulfils requirements” [17]. To be able to analyze the potential of an architecture to reach the required quality levels helps to find the problems early in the life cycle, when they are easier and

cheaper to correct than in later stages such as implementation, testing or deployment. In the case of product-line architectures (PLAs) the architecture assessment becomes crucial to ensure that the PLA is flexible enough to support different products and to allow evolution.

Quality aware implementation

The variability in functional and quality requirements must be implemented in the code. There exist lots of variation mechanisms but the overview of them is out of the scope of this paper.

Quality Testing

"Testing is an approach to validate and verify the produced artefacts. It refers to any activity that validates and verifies through the comparison of an actual result, with the result the artefact is expected to produce, based on its specifications. Deviations from the expected results are termed failures". "A failure is considered to be the result of a defect in the artefact" [39]. Quality testing is used to assure that the product has the required quality requirements. In the case of a product line, those quality requirements may have variability as well.

3. OVERVIEW OF EXISTING METHODS AND APPROACHES REGARDING QUALITY IN SOFTWARE PRODUCT LINES

In this section, an overview of methods and approaches for the previously identified tasks and practices is presented.

3.1. QUALITY VARIABILITY SPECIFICATION

In a product line, there are often products with varying levels of quality attributes. A model where quality attribute variability is modelled as well as the impacts of functional variants on quality attributes is indispensable to take the most adequate decisions during design and derivation and get the required quality levels.

Seven modelling methods that address varying quality attributes have been compared:

- Goal-based model [23]: This approach proposes to use goal-oriented analysis in product lines. Goal-oriented requirement engineering is an approach that deals with quality attributes or non-functional requirements in single systems. Two sub-models are proposed: A functional goal model and a softgoal model. Quality attributes are represented as soft-goals and the operation of those quality attributes is encoded in the functional goal sub-model as tasks. Priorities are given to each softgoal on a percentile scale to perform the analysis. And correlations are used to represent the links among functional goals and softgoals. Correlation links have different influence labels (--,-,?,+,++). Those qualitative labels are converted to quantitative values: one value for satisfiability and another for deniability.

- F-SIG (Feature-softgoal interdependency graph) [29]: The main goal of this approach is to provide a framework to record design rationale in the form of interdependencies of variable features and quality attributes. To do that a new graph is proposed: F-SIG, a union of a feature model and a SIG (Softgoal interdependency graph) [11]. In F-SIG explicit and implicit contributions from features to quality attributes are modeled. To express the degree of influence, correlations may have also a label (break:--, hurt:-, unkown: ?, Help: +, Make: ++).

- COVAMOF [50]: COVAMOF is a framework for variability modelling in software product families. With this framework it is possible to model the variability on all layers of abstraction of the product family. This framework uses the CVV (COVAMOF Variability View) which has two views: Variation Point View and Dependency View. The CVV captures variability in the product family in terms of variation points and dependencies. Quality attributes can be modelled with dependencies, a dependency can specify a property that specifies the value of a quality attribute such as performance or memory usage. Association is used to associate variation points and dependencies.

- Extended feature model [7]: It is a feature model's extension to deal with extra-functional features. A notation that extends feature models with attributes, characteristics of a feature that can be measured such as availability, cost, latency, bandwidth and relations among attributes is being proposed. Every feature may have one or more attribute relations taking a range of values in either discrete or continuous domains. It also provides automatic reasoning on those extended feature models using CSP (Constraint Satisfaction Problems).

- Definition hierarchy [35]: The hierarchy method is a logical AND tree where topmost nodes are design objectives: architectural drivers and other

quality attributes that the system is supposed to fulfil. The other nodes are design decisions and when an edge is between a design objective and a design decision it shows that this requirement is (partially) satisfied by design decisions. Each node in the definition hierarchy gets a priority that reflects the importance of that node to support the intention of its parent. Priorities are product specific; they are used as the mechanism to describe products in the definition hierarchy.

- Bayesian Belief Network: Zhang et al. [59] proposed a Bayesian Belief Network (BBN) based approach to quality prediction and assessment for a software product line. The BBN is used to explicitly model the impact of variants (especially design decisions) on system quality attributes. The feature model is used to capture functional requirements and the BBN model to capture the impact of functional variants on quality attributes.

- Quality Requirements of a Software Family (QRF) method [43] of VTT captures and maps the requirements of a product family in the family architecture by analyzing the needs of business and technology development stakeholders and the impact of these needs on the family architecture. The QRF consists of five steps: Impact analysis, which is defined through a framework that enables to define and negotiate requirements. In this step, it uses the i* framework [10], a graph called the Strategic Dependency model to define different stakeholders' functionality requirements and quality requirements; Quality analysis, where quality requirements are expressed in a way in which they can later be traced and measured. In this step the quality requirements are prioritized (low, medium, high); Variability analysis, where quality requirements that vary on the business domain are defined. This variation is specified in the Strategic Dependency model; Hierarchical domain analysis is used to map common and variable Quality Aspects to functionality (hierarchical service categories) and Quality Representation to describe architecture in a way that quality requirements can be evaluated from the architectural models. For this last step, two main means are used: Architectural styles and patterns and a NFR (Non-Functional Requirements) framework to carry out a trade-off analysis and select the style that meets the quality requirements best; and profiles to extend the architectural models to support certain quality aspects.

These modelling approaches have been analyzed to see if they cover several requirements that we consider important for modelling quality attribute variability:

- Automatic reasoning: Different reasoning tasks should be interesting: get an approximate value or level for several quality attributes starting from a set of functional requirements, detect impossible configurations starting from a set of functional and quality requirements, detect conflicts among quality attributes and provide help to perform a trade off analysis...Due to the complexity of this analysis and reasoning, it is advisable to make it automatic. To achieve automatic reasoning artificial intelligence techniques are need such as Constraint Satisfaction Problems (CSP), Boolean Satisfiability Problems (SAT) and Binary Decision Diagrams (BDD) [6].

- Quality attribute characterization: Quality attributes have vague definitions. In different domains, one quality attribute may not mean exactly the same or different names are used for the same concept. So it is necessary to specify and make quality attributes more concrete. A mechanism for describing and explaining a quality attribute adequately must be provided: A structure where a quality attribute may be explained through refinement of different levels.

- Optionality: In one product one attribute may be important and in another one this attribute may not be required. So this attribute is optional in the product line. This variability must be represented and not only at product level. It is not enough to specify this optionality when deriving products.

- Levels: Different priority levels in quality attributes are need. For example, for one family member the extensibility requirements are extremely high, whereas for others those requirements are at the lowest level. However, quality attributes are not easy to quantify due to their nature, only more concrete concepts (refinement results) may be quantified. It is necessary to provide a way to define different levels (high, medium, low) at quality attribute high level and map those levels to more concrete concerns' values.

- Quantitative and qualitative: Indirect variation must be represented with qualitative and quantitative impacts and means must be provided to quantify qualitative influences to be able to do an automatic analysis.

- Group impacts: There are some types of influential relationships that must be addressed, for instance, the influence of a group of variants. The impact of two variants together is not always the sum of the individual impacts of those two variants alone. For instance, in some applications the price of some packages that have several features or options together may be cheaper that buying all the features separately.

None of the evaluated approaches meets all the identified requirements (see Table 1). In the paper [19] can be found a more completed analysis of some of the methods.

Table 1: Evaluated methods

Requirement / Approach	Automatic reasoning	QA characterization	Optionality at PL level	Priority levels	Quantitative and Qualitative	Group impacts
Goal-based model	Yes	Yes	No	No	Yes	No
F-SIG	No	Yes	No	No	No	No
COVAMOF	Yes	No	No	No	Yes	Yes
Extended FM	Yes	No	No	No	No	No
Definition Hierarchy	No	Yes	More or less	No	Yes	No
BBN	Yes	No	No	More or less	Yes	Yes
QRF	No	More or less	Yes	Yes	No	No

3.2. QUALITY AWARE DESIGN METHODS

There are quite a lot specific methods to design product-line software architectures but not all take into account quality requirements explicitly. Some of the methods that are quality aware are: QADA [38], QUASAR [54], QASAR [9], PuLSE-DSSA [5] and SEI's PL initiative [13].

- PuLSE-DSSA: The input data for this method is a scope definition and a domain model, where the former defines the business case for the development of the product line and the latter describes common aspects and variations of applications within the product line. The output of PuLSE-DSSA is a product line architecture. The resulting architecture is evaluated according to the architecture evaluation plan. If at least some test failed, the underlying problems are examined in order to determine how the architecture development process can be continued.

- QUASAR: QUASAR is a framework that supports Quality – Driven System Architecting of product families. It is organized on three major workflows. The Preparation Workflow provides activities that support early architectural considerations. The activities of the Modelling Workflow are responsible of modelling architectural views and the variability within each view. The Evaluation Workflow includes activities to analyze the architecture consistency, variability coverage, and the achievement of qualities.

- QASAR: The Quality Attribute-oriented Software Architecture design method (QASAR) is an architecture design method that uses explicit assessment, and design for the quality requirements of a software system. It consists of two iterative processes of which the inner iteration contains three parts: functionality-based architecture design, assessment and transformation to quality requirements. Whereas the outer iteration refers to a requirement selection process. In this process a subset of requirements is selected and this subset is used for the inner iteration. QASAR does not focus on a single quality attribute but rather provides a generic method and steps for the assessment and reasoning of tradeoffs for different quality attributes.

- QADA: The QADA (Quality-driven Architecture Design and quality Analysis) method provides a systematic way to transform functional and quality requirements into software architecture. The method also uses styles and patterns as guides to carry out quality requirements in architectural descriptions with a documented design rationale.

- SEI's Product Line initiative: This approach uses the Architecture Based Design (ABD) method. This method of design addresses functional, quality and business requirements.

Only one of these methods (QADA) takes into consideration explicitly the variability in quality attributes.

3.3. ARCHITECTURE EVALUATION

In this section, software architecture evaluation methods that are related to software product lines are compared. The methods are classified in groups depending on the evaluation time and goal:

- Methods to evaluate architectures at *design phase*: Methods that are used during product line architecture design.
- Methods to evaluate *existing product-line architectures*: Methods to evaluate the software architecture of (already developed) product-lines.
- Methods to assess *variability*: The variability is a key aspect in software product lines and there are several methods that focus on this quality attribute.
- Methods oriented to evaluate *existing product architectures* to use them as basis for the product-line: Methods that are used when there exist legacy systems whose architectures can be used as a starting point for the product line.
- Methods to evaluate *both product line architecture and instantiated product architectures*: In a software product line, there are two moments where evaluation can be performed: during domain engineering and during application engineering and there are also methods that allow performing evaluations on both levels in a coordinated way.
- *Metrics*: Software product line architecture metrics to assess quality aspects.
- *Single-system architecture* evaluation methods: Methods that are not specific for product lines but can be used to evaluate product line architectures.

There are different methods to evaluate product-line architectures *in the design phase*: FAAM (Family Architecture Assessment Method) [17] to evaluate the information-system family's architectures, the QADA (Quality-driven Architecture Design and quality Analysis) approach [38] has two methods to analyse product-line architectures: The RAP (Reliability and Availability Prediction) method [26] to evaluate software reliability and availability of the architectural model and the IEE (Integrability and Extensibility Evaluation) method [44] for integrability and extensibility evaluation, REDA[1] (Reliability Evaluation of Domain Architectures) [2] to analyse the reliability of a PLA and D-SAAM (Distributed SAAM) [24], a variant of SAAM to evaluate reference architectures.

For *existing product-line architectures evaluation*: Gannod and Lutz [22] propose an approach that evaluates quality and functional requirements, Maccari [36] proposes a method to assess evolution and Riva and Rosso [49] adapt Maccari's approach.

There are some methods that *assess variability*, one of the key aspects in product-lines, at architectural-level: SBA (Scenario-Based Architecting) [1] is a method to identify and quantify the potential benefits of the different architectural variability options. There are also others that work at all layers of abstraction and not only at the software architecture: Wijnstra's approach [58] and COSVAM (The COVAMOF Software Variability Assessment Method) [15].

There are also methods oriented to evaluate *existing product architectures* to use them as basis for the product-line: SACAM (Software Architecture Comparison Analysis Method) [51] which is a method to compare architectures and Korhonen's approach [34] which analyses whether an architecture can be used as a basis for a product-line or not.

There is a method to evaluate both product line architectures and instantiated product architectures: HoPLAA [45] (Holistic Product Line Architecture Assessment), an adaptation of ATAM (Architecture Trade-off Analysis Method) for product lines. This method introduces variability in the quality attribute utility tree used for evaluation. HoPLAA addresses the requirements for the evaluation of software product line architectures in an integrated, holistic approach. It is executed in two stages; the first stage focuses on the core architecture evaluation, while the second stage targets the evaluation of individual product architectures.

There are also specific *metrics* defined for PLAs: service usage metrics [56] and Rahman's metrics [47].

There are some very popular architecture evaluation methods that can be also used to evaluate product-line architectures like SAAM (Software Architecture Analysis Method) [12] and its variants. And the successor of SAAM: ATAM (Architecture Trade-off Analysis Method) [12]. These methods are not product-line specific, they are used to evaluate single-product architectures but they are adequate to address qualities that are *product-line quality attributes* such as maintainability and extensibility among others. ATAM has been used in product-line contexts [20] [21], even though there is no special treatment in ATAM for product-line architectures; in these case studies the product-line particular aspects are addressed implicitly as quality requirements.

In the next table, a summary of the performed comparison is presented. In [18] a more completed survey of most of the methods can be found.

[1] This abbreviated name is not original, it is used for convenience

Table 2: Comparison of evaluation methods

Evaluation Method	Goal	Attribute Types	Evaluation Techniques	Process Description	Method's validation	Relation with other methods
FAAM (Family Architecture Assessment Method)	Stakeholder oriented assessment of information-system family's architectures	**PL qualities**: Interoperability, extensibility…	Scenarios, other techniques	Very detailed: Steps, guidelines, roles…	2 case studies in different domains	Extends SAAM
REDA (Reliability Evaluation of Domain Architectures)	Evaluate PLAs to predict reliability	**Domain qualities**: Reliability	Failure cases, qualitative reliability model (QlRM), metrics…	Reasonable: Steps, techniques…	Case study in automotive control systems	-
D-SAAM (Distributed SAAM)	Evaluate reference architectures reducing the organisational impact	**PL qualities**: Maintainability	Scenarios	Well explained: Steps, guidelines, roles…	Applied on a copier systems PLA	Variant of SAAM
RAP	Method for evaluating software reliability and availability from the architectural mode	**Domain qualities**: Reliability and availability	Markov chain model, simulations, estimations	Well explained: Steps, guidelines…	One case example	-
IEE	Method for integrability and extensibility evaluation.	**PL qualities**: Integrability and extensibility	Scenarios	Reasonable: Steps, guidelines…	One case example	-
Gannod and Lutz's approach	Analyse an existing PLA	**PL qualities**: Modifiability **Common behaviour**	Scenarios and model checking	Reasonable: Steps, guidelines…	Applied on a telescopes PL	Includes a step similar to SAAM
Maccari's approach	Assess the capability of a PLA to adapt to evolution	**PL qualities**: Evolution related ones: Scalability, modifiability…	Scenarios	Briefly explained but illustrated through case studies	2 case studies in different domains	-
Riva and Rosso's approach	Assess PLAs for evolution	**PL qualities**: Flexibility, modifiability	Scenarios, experience-based analysis	Explained with a case study	Case study in a mobile terminals PLA	Adapts Maccari's approach
SBA (Scenario-Based Architecting)	Identify and quantify the benefits of different variability options	**PL qualities**: Variability	Scenario-based quantitative analysis	Well explained: Steps, guidelines…	2 case studies of the medical domain	Uses SQUASH [52]
COSVAM (The COVAMOF Software Variability Assessment Method)	Evaluate the variability of a PL in a evolution context	**PL qualities**: Variability	Product scenarios, expert-based analysis…	Well explained: Steps, guidelines…	Applied on an intelligent traffic systems PL	-
Wijnstra's approach	Assess a PL for the way it deals with variation	**PL qualities**: Variability	Study the gathered information	An overview	Case study in the medical domain	-
SACAM (Software Architecture Comparison Analysis Method)	Compare candidate architectures (existing product architectures)	**PL qualities** **Domain qualities**	Scenarios, tactics and metrics	Detailed explanation: Steps, guidelines, participants…	An example to illustrate the method	-
Korhonen's approach	Assess system adaptability to a product family	**PL qualities**: Adaptability, configurability… **Domain qualities**: Reliability, performance…	Scenarios	Explained with a case study	Applied on a case study of mobile machines	Loosely based on SAAM and ATAM
HoPLAA	Evaluation of software product line architectures in an integrated, holistic approach	**PL qualities** **Domain qualities**	Scenarios	Comprehensively explained	A example	Adaptation of ATAM
Service Utilization metrics	Assess and improve PLAs	**PL qualities**: Structural soundness	Metrics	Comprehensively explained	Case study in a digital library PLA	-
Rahman's metrics	Measure the quality attributes of a PLA	**PL qualities**: Reusability, modularity	Metrics	Reasonable	Case study in a library system	Include Service Utilization metrics

3.4. TESTING

There are several approaches for software product lines testing: Nebut at al. [41] defines an approach to make the test generation automatic, ScenTED (Scenario-based Test case Derivation) [46] approach also facilitates the derivation of test cases from use cases, *McGregor* [39] also presents some testing practices for product lines, PLUTO [8] testing methodology, testing tools for product lines such as RITA [30], etc. However, most of the approaches focus on functional requirement's testing and non-functional requirements are not addressed explicitly. To the best of our knowledge, there is only one product line testing technique that addresses a quality attribute (performance) [48].

4. CONCLUSIONS AND REQUIREMENTS FOR QUALITY DRIVEN DOMAIN ENGINEERING

Modelling

No modelling approach meets all the identified requirements. As a conclusion a new approach or an extension of an existing approach could be interesting to address all the identified requirements to model variable quality attributes.

Quality aware design

Only one of the methods considers explicitly the variability in quality requirements.

Evaluation

Among the surveyed evaluation methods, most of them focus on evaluating *product-line quality attributes* (flexibility) at product-line architecture level and there are few methods to evaluate execution or *domain-relevant quality attributes* (performance, reliability…). And there is only one method to evaluate both architectures in a holistic way.

However, there are (no product-line specific) single-system architecture evaluation methods that can be used for derived product architectures and also for PLAs. Single-product architecture evaluation is quite a mature field where a lot of research has been done and techniques and methods developed.

Regarding product line architecture evaluation, there is an issue that no method answers and that is worth mentioning. To assess all the instances of the product-line may not be worthwhile due to the high cost. However, it is possible to shorten product-architecture evaluations because the product architecture evaluation is a variation of the product-line architecture evaluation. But, as far as we know, no way has been provided to reduce the number of evaluations in a cost-effective way while evaluating the whole line.

Testing

There is only one software product line testing technique that addresses a quality attribute.

In general, quality in software product lines is an area in which more research would be welcome. About the studied approaches, QADA is one of the most complete approaches, it covers most of the identified tasks: it includes a modelling approach; QRF, a design method and evaluation methods RAP and IEE. However, QADA has also its inconveniences as it does not cover all the requirements for modelling quality variability and it is quite complicated to apply.

5. RELATED WORK

In this paper, several method comparisons that can help to assess quality aspects in software product lines are presented.

There are many surveys related to software product lines, for instance, [14] presents a survey of software reuse processes, where several product line processes are compared and [37] presents an overview of software product line architecture design methods.

There are also some survey papers of single-system software architecture evaluation methods: [16], a comparison of scenario based methods [28][3], survey of methods for reliability and availability evaluation [27]. And also one of our previous works, a survey of specific software architecture methods for software product lines [18]. There also are surveys on software product line testing such as [53].

However, none of the works provide a global view of approaches and methods to address and assess quality attributes during all the development phase.

6. FUTURE WORK

We are working on developing a method and tool support for facilitating quality aware software product line engineering. The method is based on a variability model where functional and quality variability are modelled. This method is used to facilitate quality assessment and management during software product line engineering and also to allow quality aware derivation of products.

Quality assessment in software product line, has a high cost if all the instances of the product-line are assessed and there are few methods to validate operational qualities in software product lines that take into account variability. However, quality variability (the variability model) may provide useful information about which products should be validated. This way it is possible to focus on representative products and reduce the number of validations in a cost-effective way instead of validating all the products. Quality validation can be performed at different stages: during design (software architecture evaluation) or after implementation (testing). Using our method, it is possible to use validation methods for single systems, which is a quite a mature field, where a lot of research techniques and methods have been developed.

ACKNOWLEDGEMENTS

This work was partially supported by The Basque Government's Department of Education, universities and research. Leire Etxeberria enjoys a doctoral grant of the researchers' formation program.

REFERENCES

[1] P. America, D. K. Hammer, M. T. Ionita, J. H. Obbink, and E. Rommes. Scenario-based decision making for architectural variability in product families. *Software Process: Improvement and Practice*, 10(2):171–187, 2005.

[2] M. Auerswald, M. Herrmann, S. Kowalewski, and V. Schulte-Coerne. Reliability-oriented product line engineering of embedded systems. In *PFE '01: Revised Papers from the 4th International Workshop on Software Product-Family Engineering*, pages 83–100, London, UK, 2002. Springer-Verlag.

[3] M. A. Babar and I. Gorton. Comparison of scenario-based software architecture evaluation methods. In *APSEC '04: Proceedings of the 11th Asia-Pacific Software Engineering Conference (APSEC'04)*, pages 600–607, Washington, DC, USA, 2004. IEEE Computer Society.

[4] L. Bass, P. Clements, and R. Kazman. *Software architecture in practice*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1998.

[5] J. Bayer, O. Flege, and C. Gacek. Creating product line architectures. In F. van der Linden, editor, *Software Architectures for Product Families, International Workshop IW-SAPF-3*, volume 1951 of *Lecture Notes in Computer Science*, pages 210–216. Springer, 2000.

[6] D. Benavides, S. Segura, P. Trinidad, and A. Ruiz-Cortés. A first step towards a framework for the automated analysis of feature models. In *SPLC, Software Product Line Conference*, 2006.

[7] D. Benavides, P. Trinidad, and A. Ruiz-Cortés. Automated reasoning on feature models. In O. Pastor and J. F. e Cunha, editors, *Advanced Information Systems Engineering, 17th International Conference, CAiSE, Proceedings*, volume 3520 of *Lecture Notes in Computer Science*, pages 491–503. Springer, 2005.

[8] A. Bertolino and S. Gnesi. Pluto: A test methodology for product families. In van der Linden PFE2003, pages 181–197.

[9] J. Bosch. *Design and use of software architectures: adopting and evolving a product-line approach*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 2000.

[10] L. Chung, D. Gross, and E. S. K. Yu. Architectural design to meet stakeholder requirements. In *WICSA1: Proceedings of the TC2 First Working IFIP Conference on Software Architecture (WICSA1)*, pages 545–564, Deventer, The Netherlands, The Netherlands, 1999. Kluwer, B.V.

[11] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos. *Non-Functional Requirements in Software Engineering (The Kluwer International Series in Software Engineering Volume 5)*. Springer, October 1999.

[12] P. Clements, R. Kazman, and M. Klein. *Evaluating Software Architectures: Methods and Case Studies*. Addison-Wesley Professional, January 2002.

[13] P. Clements, L. Northrop, and L. M. Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley Professional, August 2001.

[14] E. S. de Almeida, A. Alvaro, D. Lucrédio, V. C. Garcia, and S. R. de Lemos Meira. A survey on software reuse processes. In D. Zhang, T. M. Khoshgoftaar, and M.-L. Shyu, editors, *IRI*, pages

66–71. IEEE Systems, Man, and Cybernetics Society, 2005.

[15] S. Deelstra, M. Sinnema, J. Nijhuis, and J. Bosch. Cosvam: A technique for assessing software variability in software product families. In *ICSM '04: Proceedings of the 20th IEEE International Conference on Software Maintenance*, pages 458–462, Washington, DC, USA, 2004. IEEE Computer Society.

[16] L. Dobrica and E. Niemela;. A survey on software architecture analysis methods. *IEEE Trans. Softw. Eng.*, 28(7):638–653, 2002.

[17] T. J. Dolan. *Architecture Assessment of Information-System Families: a practical perspective*. PhD thesis, Tech. Univ. Eindhoven, Netherlands, 2001.

[18] L. Etxeberria and G. Sagardui. Product-line architecture: New issues for evaluation. In J. H. Obbink and K. Pohl, editors, *9th International Conference on Software Product Lines, SPLC, Proceedings*, volume 3714 of *Lectures Notes in Computer Science*, pages 174–185. Springer, 2005.

[19] L. Etxeberria, G. Sagardui, and L. Belategi. Modelling variation in quality attributes. In K. Pohl, P. Heymans, K.-C. Kang, and A. Metzger, editors, *First International Workshop on Variability of Software-Intensive Systems (VaMos 2007)*, volume Lero Technical report 2007-1. Lero, 2007.

[20] S. Ferber, P. Heidl, and P. Lutz. Reviewing product line architectures: Experience report of atam in an automotive context. In *PFE '01: Revised Papers from the 4th International Workshop on Software Product-Family Engineering*, pages 364–382, London, UK, 2002. Springer-Verlag.

[21] B. P. Gallagher. Using the architecture tradeoff analysis method to evaluate a reference architecture: A case study. Technical Report CMU/SEI-2000-TN-007, SEI, 2000.

[22] G. C. Gannod and R. R. Lutz. An approach to architectural analysis of product lines. In *ICSE '00: Proceedings of the 22nd international conference on Software engineering*, pages 548–557, New York, NY, USA, 2000. ACM Press.

[23] B. González-Baixauli, J. C. S. do Prado Leite, and J. Mylopoulos. Visual variability analysis for goal models. In *12th IEEE International Conference on Requirements Engineering (RE)*, pages 198–207. IEEE Computer Society, 2004.

[24] B. Graaf, H. van Dijk, and A. van Deursen. Evaluating an embedded software reference architecture " industrial experience report ". In *CSMR '05: Proceedings of the Ninth European Conference on Software Maintenance and Reengineering*, pages 354–363, Washington, DC, USA, 2005. IEEE Computer Society.

[25] IEEE. Ieee standard 1061-1992. ieee standard for a software quality metrics methodology, 1993.

[26] A. Immonen. *Software Product Lines, Research Issues in Engineering and Management*, chapter A Method for Predicting Reliability and Availability at the Architecture Level, pages 373–422. Springer, 2006.

[27] A. Immonen and E. Niemelä. Survey of reliability and availability prediction methods from the viewpoint of software architecture. *Softw Syst Model (2008)*, 7:49–65, 2008.

[28] M. T. Ionita, D. K. Hammer, and H. Obbink. Scenario-based software architecture evaluation. In *Methods: An Overview, Workshop on Methods and Techniques for Software Architecture Review and Assessment at the International Conference on Software Engineering, Orlando, Florida, USA*, May 2002.

[29] S. Jarzabek, B. Yang, and S. Yoeun. Addressing quality attributes in domain analysis for product lines. *IEE Proceedings - Software*, 153(2):61–73, 2006.

[30] R. Kauppinen and J. Taina. Rita environment for testing framework-based software product lines. In P. Kilpeläinen and N. Päivinen, editors, *Proceedings of the Eighth Symposium on Programming Languages and Software Tools (SPLST)*, pages 58–69. University of Kuopio, Department of Computer Science, 2003.

[31] R. Kolb, J. D. McGregor, and D. Muthig, editors. *First International Workshop on Quality Assurance in Reuse Contexts (QUARC)*, IESE-Report No. 096.04/E. Fraunhofer IESE, August 2004.

[32] R. Kolb, J. D. McGregor, and D. Muthig. Introduction to quality assurance in reuse contexts. In *First International Workshop on Quality Assurance in Reuse Contexts (QUARC)*, 2004.

[33] R. Kolb and D. Muthig, editors. *First eWorkshop on Quality Assurance for Software Product Lines: Strategic Issues*, IESE-Report No. 013.05/E. Fraunhofer IESE, January 2005.

[34] M. Korhonen and T. Mikkonen. Assessing systems adaptability to a product family. *J. Syst. Archit.*, 50(7):383–392, 2004.

[35] J. Kuusela and J. Savolainen. Requirements engineering for product families. In *ICSE '00: Proceedings of the 22nd international conference on Software engineering*, pages 61–69, New York, NY, USA, 2000. ACM Press.

[36] A. Maccari. Experiences in assessing product family software architecture for evolution. In *ICSE '02: Proceedings of the 24th International Conference on Software Engineering*, pages 585–592, New York, NY, USA, 2002. ACM Press.

[37] M. Matinlassi. Comparison of software product line architecture design methods: Copa, fast, form, kobra and qada. In *26th International Conference on Software Engineering (ICSE)*, pages 127–136. IEEE Computer Society, 2004.

[38] M. Matinlassi, E. Niemelä, and L. Dobrica. Quality-driven architecture design and quality analysis method: A revolutionary initiation approach to a product line architecture. Technical Report VTT-PUBS-456, VTT Electronics, jan 2002.

[39] J. D. McGregor. Testing a software product line. Technical Report CMU/SEI-2001-TR-022, SEI, dec 2001.

[40] V. Myllärniemi, T. Männistö, and M. Raatikainen. Quality attribute variability within a software product family architecture. In *Second International conference on Quality of Software Architecture QoSA*, 2006.

[41] C. Nebut, Y. L. Traon, and J.-M. Jézéquel. *Software Product Lines, Research Issues in Engineering and Management*, chapter System Testing of Product Lines: From Requirements to Test Cases, pages 447–477. Springer, 2006.

[42] E. Niemelä. Architecture centric software family engineering, product family engineering seminar. Tutorial in 5th Working IEEE/IFIP Conference on Software Architecture (WICSA), 2005.

[43] E. Niemelä. Quality driven family architecture development. Tutorial in SPLC (Software Product Line Conference), 2005.

[44] E. Niemelä and M. Matinlassi. Quality evaluation by qada. Tutorial in 5th Working IEEE/IFIP Conference on Software Architecture (WICSA), 2005.

[45] F. G. Olumofin and V. B. Misic. Extending the atam architecture evaluation to product line architectures. In *WICSA '05: Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05)*, pages 45–56, Washington, DC, USA, 2005. IEEE Computer Society.

[46] K. Pohl and A. Metzger. Software product line testing. *Commun. ACM*, 49(12):78–81, 2006.

[47] A. Rahman. Metrics for the structural assessment of product line architecture. Master's thesis, School of Engineering, Blekinge Institute of Technology, 2004.

[48] S. Reis, A. Metzger, and klaus Pohl. A reuse technique for performance testing of software product lines. In P. Knauber, C. Krueger, and T. Trew, editors, *SPLIT 2006 - Third International Workshop on Software Product Line Testing*, volume Computer Science Reports. Mannheim University of Applied Sciences - Computer Science Department, 2006.

[49] C. Riva and C. D. Rosso. Experiences with software product family evolution. In *IWPSE '03: Proceedings of the 6th International Workshop on Principles of Software Evolution*, page 161, Washington, DC, USA, 2003. IEEE Computer Society.

[50] M. Sinnema, S. Deelstra, J. Nijhuis, and J. Bosch. Covamof: A framework for modeling variability in software product families. In R. L. Nord, editor, *3rd International Conference on Software Product Lines, SPLC, Proceedings*, volume 3154 of *Lecture Notes in Computer Science*, pages 197–213. Springer, sep 2004.

[51] C. Stoermer, F. Bachmann, and C. Verhoef. Sacam: The software architecture comparison analysis method. Technical Report CMU/SEI-2003-TR-006, SEI, 2003.

[52] M. Svahnberg, C. Wohlin, L. Lundberg, and M. Mattsson. A quality-driven decision-support method for identifying software architecture candidates. *International Journal of Software Engineering and Knowledge Engineering*, 13(5):547–573, 2003.

[53] A. Tevanlinna, J. Taina, and R. Kauppinen. Product family testing: a survey. *SIGSOFT Softw. Eng. Notes*, 29(2):12–12, 2004.

[54] S. Thiel. On the definition of a framework for an architecting process supporting product family development. In *PFE '01: Revised Papers from the 4th International Workshop on Software Product-Family Engineering*, pages 125–142, London, UK, 2002. Springer-Verlag.

[55] S. Thiel and A. Hein. Systematic integration of variability into product line architecture design. In *SPLC 2: Proceedings of the Second International Conference on Software Product Lines*, pages 130–153, London, UK, 2002. Springer-Verlag.

[56] A. van der Hoek, E. Dincel, and N. Medvidovic. Using service utilization metrics to assess the structure of product line architectures. In *METRICS '03: Proceedings of the 9th International Symposium on Software Metrics*, page 298, Washington, DC, USA, 2003. IEEE Computer Society.

[57] F. van der Linden, editor. *Software Product-Family Engineering, 5th International Workshop, PFE 2003, Siena, Italy, November 4-6, 2003, Revised Papers*, volume 3014 of *Lecture Notes in Computer Science*. Springer, 2004.

[58] J. G. Wijnstra. Evolving a product family in a changing context. In van der Linden PFE2003, pages 111–128.

[59] H. Zhang, S. Jarzabek, and B. Yang. Quality prediction and assessment for product lines. In J. Eder and M. Missikoff, editors, *15th International Conference on Advanced Information Systems Engineering, CAiSE, Proceedings*, volume 2681 of *Lecture Notes in Computer Science*, pages 681–695. Springer, 2003.