Defining and Validating a Multimodel Approach for Product Architecture Derivation and Improvement

Javier González-Huerta, Emilio Insfrán, and Silvia Abrahão

ISSI Research Group, Universitat Politècnica de València
Camino de Vera, s/n, 46022, Valencia, Spain
{jagonzalez,einsfran,sabrahao}@dsic.upv.es

Abstract. Software architectures are the key to achieving the non-functional requirements (NFRs) in any software project. In software product line (SPL) development, it is crucial to identify whether the NFRs for a specific product can be attained with the built-in architectural variation mechanisms of the product line architecture, or whether additional architectural transformations are required. This paper presents a multimodel approach for quality-driven product architecture derivation and improvement (QuaDAI). A controlled experiment is also presented with the objective of comparing the effectiveness, efficiency, perceived ease of use, intention to use and perceived usefulness with regard to participants using QuaDAI as opposed to the Architecture Tradeoff Analysis Method (ATAM). The results show that QuaDAI is more efficient and perceived as easier to use than ATAM, from the perspective of novice software architecture evaluators. However, the other variables were not found to be statistically significant. Further replications are needed to obtain more conclusive results.

Keywords: Software Product Lines, Architectural Patterns, Quality Attributes, Model Transformations, Controlled Experiment.

1 Introduction

The quality attributes of a software system (e.g., performance, modifiability, and availability) are, to a great extent, permitted or precluded by its architecture [9]. In the case of Software Product Line (SPL) development, in which a set of software-intensive systems sharing a common set of features are developed by taking advantage of the massive reuse of software assets, the product line architecture should have variation mechanisms that help to achieve a set of explicitly allowed variations [9]. These variations may include structural, behavioral and of course quality concerns. The product line architecture should therefore be designed to cover the whole set of variations within the product line. The product architecture can thus be derived from the product line architecture by exercising its built-in architectural variation mechanisms, which support both the functional and non-functional requirements[1] (NFRs) for a specific product.

[1] Non-Functional Requirements can be defined as the qualities that a product must have, such as an appearance, or a property of speed or accuracy [30].

A. Moreira et al. (Eds.): MODELS 2013, LNCS 8107, pp. 388–404, 2013.

Once it has been derived, the product architecture should be evaluated in order to guarantee that it meets the specific requirements of the product under development [9]. However, in those cases in which levels of quality attributes that fall outside the original specification of the product line are needed (and cannot be attained by using product line variation mechanisms), certain architectural transformations may be applied to the product architecture to ensure that these NFRs are met [5].

Although several methods for architecture derivation and improvement in SPL development have been proposed over the last few years (e.g., [23], [28], [19], [31], [6], [8], [29]), there is still a need for approaches that model the impact between architectural design decisions and quality attributes and use this information to enhance the quality attribute levels of product architectures. We have addressed this problem, in previous works [17] [18] [20], by proposing an approach with which to ensure the desired quality attribute levels for a product by applying architectural transformations to a product architecture derived from a product line architecture using a multimodel. This multimodel represents a set of interrelated viewpoints of the product line and the semantic relationships among elements in each viewpoint. It also allows the product line architecture, the metrics for its evaluation and the relationships among architectural transformations and NFRs to be represented.

In this paper, we present the quality-driven product architecture derivation and improvement (QuaDAI) method, which uses the multimodel to guide the software architect in the derivation, evaluation and improvement of product architectures in a model-driven software product line development process. Since in the software architecture field there is a lack of empirical evidences that support the claimed benefits and capabilities of methods, techniques and tools [1], we also present the results of its empirical validation through a controlled experiment. The objective of this paper is the following: i) to present a method, consisting of a set of activities carried out by model transformation processes, thus allowing us to derive product architectures from the product line architecture, to evaluate the product architecture obtained and, when required, to improve the architectures' quality attribute levels by applying pattern-based architectural transformations; and ii) to evaluate the effectiveness, efficiency, perceived ease of use, usefulness and intention to use of the method in comparison with the Architecture Trade-Off Analysis Method (ATAM) [22]. This evaluation was done by conducting a controlled experiment with fifth year Computer Science students.

The remainder of the paper is structured as follows. Section 2 discusses existing approaches that deal with the derivation, evaluation and improvement of software architectures when following a product line approach. Section 3 presents our multimodel approach for the derivation, evaluation and improvement of product architectures with the desired quality attributes. Section 4 presents the preliminary results of the validation of the approach through a controlled experiment. Finally, the conclusions and future work are presented in Section 5.

2 Related Work

Several approaches for the quality evaluation and analysis of SPL architectures have been proposed over the last few years (e.g., [23], [28], [19], [31]). Among them, Kim et al. [23] and Olumofin and Misic [28] propose two extensions of ATAM (i.e., EATAM [23] and HoPLAA [28]) with which to assess the quality of both product line and product architectures. Both methods extend ATAM with the qualitative analytical treatment of variation points.. Although HoPLAA and EATAM consider the architectural variation points during the architecture design, they lack a systematic mechanism that can be used to deal with those cases in which the NFRs of the product under development are not within the range of values permitted by the architectural variability. In addition, they do not explicitly represent the relationships between the architectural improvements and the quality attributes. These relationships could be reused during the application engineering stage each time a new product architecture needs to be improved, thus facilitating the evaluator task. Neither EATAM nor HoPLAA have been empirically validated. HoPLAA had been compared with ATAM in a running example and the validation of EATAM has not yet taken place.

Guana and Correal [19] proposed an approach that generates an evaluation report with the possible architectural configuration that meets the required quality attributes of the product under development. They defined relationships between a variability feature tree and the functional components, and associated these relationships with a quality scenario, which is analyzed at evaluation time. Roos-Frantz et al. [31] present an approach that automates the quality analysis of SPLs. This automation is performed by associating quality information with the variability view (expressed by means of orthogonal variability models), and by using constraint programming to perform the analysis tasks. The problem is partially addressed by the approaches presented in both [31] and [19]. They explicitly define the relationships amongst system views and use the information to predict the quality attribute levels of the product under development. However, they do not provide mechanisms to measure whether these quality attribute levels are present in the software artifacts. These approaches can also predict the quality attributes of a configuration, but they cannot deal with products with quality attribute levels that are not allowed by the product architecture variation mechanisms. With regard to validation, the approach in [31] has been theoretically but not empirically validated.

Several other approaches deal with the automatic derivation of product architectures in SPL development (e.g., [6], [8], [29]). In the approach by Botterweck et al. [6], the product architectures are produced by means of an ATL model transformation process, which takes as input a domain architecture model and an application feature model and generates an application architecture model, by simply copying the software components. Similarly, Cabello et al. [8] produce application architectural models by means of a QVT transformation. The transformation takes as input the variability view expressed in a feature model and the modular view of the architecture, and generates the PRISMA component and connector architectural view. Finally, Perovich et al. [29], automate the derivation of product architectures by taking as input a feature configuration model. The transformation encapsulates the

knowledge of how to build the product architecture when the corresponding feature is present in the feature configuration model. However, when deriving the product architecture these approaches do not take into account quality attribute requirements and they do not consider the application of patterns or architectural transformations to improve the product architectures obtained. None of the aforementioned approaches has been empirically validated.

In summary, there is a need for empirically validated approaches that model the impact between architectural design decisions and quality attributes and use this information to derive the product architectures and to evaluate and enhance their quality attribute levels. The use of the multimodel in these tasks allows the knowledge to be reused in order to facilitate the evaluation tasks, providing mechanisms that automate the selection of the architectural transformations that best fit the NFRs.

3 QuaDAI: Architecture Derivation and Improvement

QuaDAI is a method for the derivation, evaluation and improvement of product architecture that defines an artifact (the multimodel) and a process consisting of a set of activities conducted by model transformations. QuaDAI relies on a multimodel [17] that allows the explicit representation of different viewpoints of a software product line and the relationships among them.

3.1 A Multimodel for Specifying SPLs

A multimodel is a set of interrelated models that represents the different viewpoints of a particular system. A viewpoint is an abstraction that yields the specification of the whole system restricted to a particular set of concerns and it is created with a specific purpose in mind. In any given viewpoint it is possible to make a model of the system that contains only the objects that are visible from that viewpoint [4]. Such a model is known as a viewpoint model, or a view of the system from that viewpoint. The multimodel also allows the definition of relationships among model elements in those viewpoints, which captures the missing information that the separation of concerns could lead to. The multimodel can be used for the specification of single systems, families of systems and in this work is used for the representation of an SPL. The multimodel plays two different roles in SPL development: i) in the *domain engineering phase,* in which the core asset base is created, the multimodel explicitly represents the different viewpoints of the SPL and the relationships among these views; ii) in the *application engineering phase,* in which the final product is derived, the relationships drive the different model transformation processes that constitute the production plan used to produce the final product. The concepts introduced in this section are illustrated through the use of a running example: a software product line from the automotive domain which comprises the safety-critical embedded software systems responsible for controlling a car.

The multimodel used to specify SPLs is composed of (at least) four interrelated viewpoints: *functional*, *variability*, *quality*, and *transformation*:

- The **variability viewpoint** expresses the commonalities and variability within the product line. Its main element is the feature, which is a user-visible aspect or characteristic of a system [9]. The variability view of the multimodel has been defined using a variant [11] of the cardinality-based feature model [16], defined specifically for application in a model-driven product line development context (see Fig. 1 top left).
- The **functional viewpoint** expresses the structure of a family of systems represented by the SPL architecture and the core assets (e.g., software components) that satisfy the requirements of the different features (see Fig. 1 top right). The functional view has been defined using the Architectural Analysis and Design Language (AADL) [15]. AADL defines a textual and graphical representation of the runtime architecture of software systems as a component-based model in terms of tasks, their interactions and the hardware platform on which the systems are executed.
- The **quality viewpoint** expresses the decomposition of quality characteristics for SPL into sub-characteristics, quality attributes, and metrics as well as the impacts and constraints among quality attributes. It is represented by a quality model for software product lines [18]. This model extends the ISO/IEC 25010 (SQuaRE) standard [21], thus providing the quality assurance and evaluation activities in SPL development (see Fig. 1 bottom left). The multimodel also allows the specification of product line NFRs as constraints defined over the quality model, affecting characteristics, sub-characteristics and quality attributes [17]. The definition of NFRs as constraints in the quality model provides a mechanism for the automatic validation of their fulfillment once the software artifacts have been obtained.
- The **transformation viewpoint** contains the explicit representation of the design decisions realized by the different model transformation processes that integrate the production plan for a model-driven SPL (see Fig. 1 bottom right). Alternatives appear in a model transformation process when a set of constructs in the source model admits different representations in the target model. The application of each alternative transformation could generate alternative target models that may have the same functionality but might differ in their quality attributes. In this work, we focus on architectural patterns [14], [25]. Architectural patterns specify solutions to recurrent problems that occur in specific contexts [7]. They also specify how the system will deal with one aspect of its functionality, impacting directly on the product quality attributes. Architectural patterns can be represented as architectural transformations, as a means to ensure the quality attributes attained by the product architectures.

In addition to the viewpoints, the multimodel also allows the definition of relationships among elements on each viewpoint with different semantics such as composition, impact or constraint relationships [17]. The composition relationship allows a model element A in a viewpoint to be decomposed into elements B, C… in other viewpoints. The impact relationship allows a model element *A* in a viewpoint impact on an element *B* in other viewpoint (e.g., an entity in a viewpoint impacts positively or negatively on a quality attribute from the quality viewpoint). These impact relationships may require additional attributes in which to store their quantification. Finally, constraint relationships allow more complex relationships at multimodel level to be expressed using an OCL-like syntax.

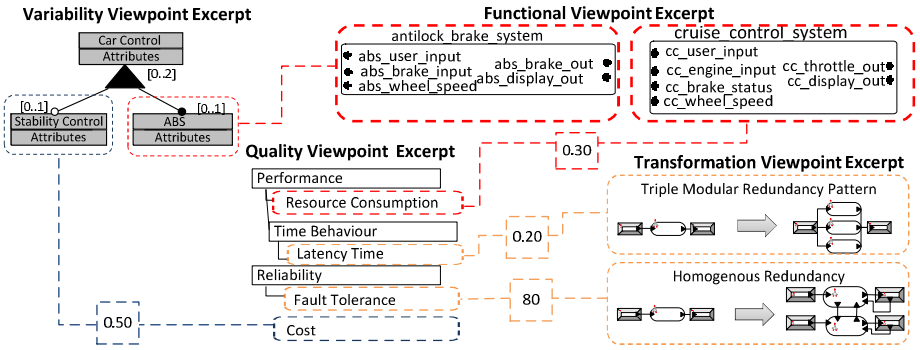

Fig. 1. SPL multimodel overview

In particular, the following types of relationships among elements in the different viewpoints can be defined in the multimodel:

- **Composition relationship:** A composition relationship can be defined between elements in the functional and variability viewpoints. A set of elements in the functional viewpoint can be combined in order to fulfill the requirements of one or more features (in Fig. 1 the *ABS* feature in a car is fulfilled by the *antilock_braking_system* component).
- **Impact relationship:** A composition relationship can be defined between elements in the transformation and quality viewpoints. The selection of a particular transformation in the transformation viewpoint may affect one or more NFRs defined over the quality model (in Fig. 1 the application of the *Homogenous Redundancy pattern* impacts positively on the product fault tolerance). A domain expert therefore establishes the relationship among alternative transformation and quality attributes by determining how a given transformation supports a given quality attribute, based on empirical evidence or on his/her experience. This tradeoff analysis is performed by applying the Analytic Hierarchy Process (AHP) [32]. AHP is a decision-making technique used to resolve conflicts in which it is necessary to address multi-criteria comparisons. The result of the AHP is a weight that shows the relative support of an alternative with regard to a given quality attribute, and it is stored in the quantification attributes of the impact relationship (e.g., in Fig. 1, the *triple modular redundancy pattern* supports *latency time* with a relative weight of 0.20).

On the one hand, the relationships among the functional, variability, and quality viewpoints can be used to drive the product configuration, the core asset selection and the product architecture derivation processes. On the other hand, the relationships defined between the transformation and quality viewpoints allow the use of the quality attributes as a decision factor when choosing from alternative pattern-based architectural transformations.

3.2 QuaDAI Process

The QuaDAI process includes different activities in which the multimodel is used to drive the model transformation processes for the derivation, evaluation and improvement of product architectures in SPL development. The activity diagram of the process supporting the approach is shown in Fig. 2 (a). It consists of the product architecture derivation from the product line architecture in the *Product Architecture Derivation* activity, its evaluation using the *Product Architecture Evaluation* activity and, in those cases in which the NFRs cannot be attained, its transformation through the application of pattern-based architectural transformations in the *Product Architecture Transformation* activity. Once this latter activity has been carried out, the resulting architecture must be evaluated again using the *Product Architecture Evaluation Activity.*

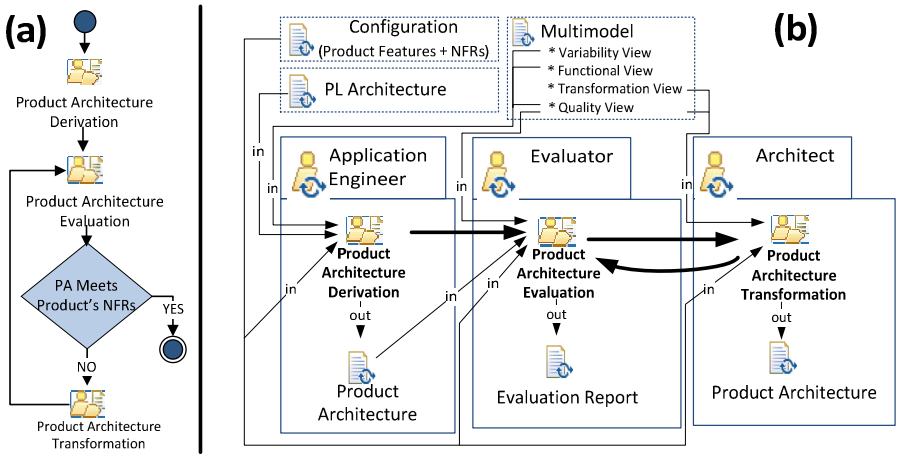

Fig. 2. Overview of the QuaDAI process

Product Architecture Derivation. The product architecture is derived from the product line architecture in the *Product Architecture Derivation* activity, taking as input the product line architecture, the variability and functional viewpoints of the multimodel, and the product configuration, containing both the product specific features and the product-specific NFRs selected by the application engineer (see Fig. 2(b)). In this activity, the decision as to which functional components should be deployed in the product architecture is made by considering the following: i) the composition relationships between features and functional components; ii) the impact relationships between functional components and NFRs; and iii) the impact relationships between features and NFRs. The output of this activity is a first version of the product architecture which must be evaluated in order to analyze the attainment of non-functional requirements.

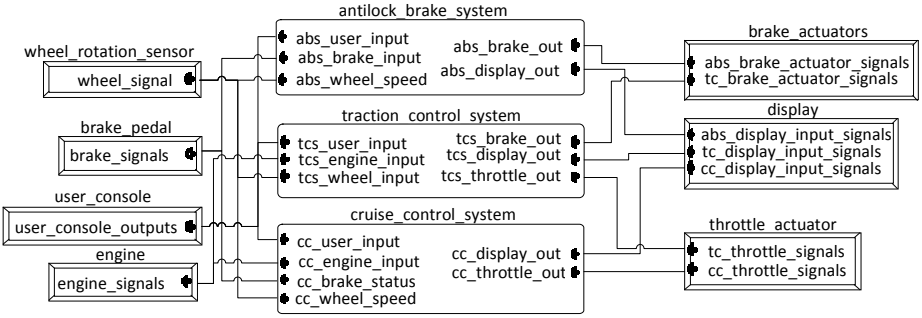

Fig. 3. Excerpt of the Product Line Architecture

Fig. 4 shows the product architecture generated by the product architecture derived from the product line architecture (shown in Fig. 3) for the automotive example when the application engineer selects only the ABS feature and introduces the product specific NFRs, which come from the system's requirements, demanding a fault tolerance of the ABS greater than 99.5% and restricting the ABS latency time to 5ms.

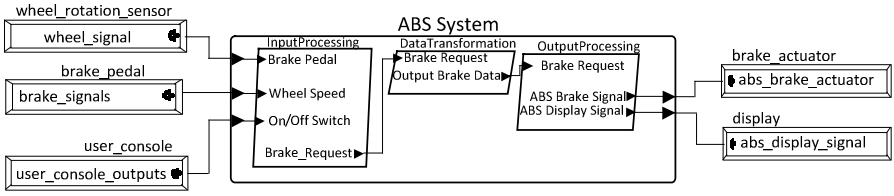

Fig. 4. Portion of the Product Architecture showing the ABS system

Product Architecture Evaluation. In the second model transformation process, the *Product Architecture Evaluation* applies the software measures contained in the quality viewpoint of the multimodel to a product architecture in order to evaluate whether or not it satisfies the desired NFRs. This transformation takes as input the product architecture derived, the product specific NFRs and the quality viewpoint of the multimodel (quality model) containing the metrics to be applied in order to measure the NFRs, generating as output an evaluation report (see Fig. 2(b)). The evaluation for the example architecture shown in Fig. 4 may conclude that the architecture meets the latency NFR but that the fault tolerance NFR is not achieved, and architectural transformations may thus be required.

Product Architecture Transformation. Finally, in those cases in which the non-functional requirements cannot be achieved by exercising the architectural variability mechanisms, in the third activity, the *Product Architecture Transformation* applies pattern-based architectural transformations to the product architecture. The inputs for this activity are the product architecture, the relative importance of the different NFRs and the transformation viewpoint of the multimodel, containing the representation of the transformations to be applied. It generates a product architecture as output in an

attempt to cover the NFRs prioritized by the architect (see Fig. 2(b)). The architect introduces the relative importance of each NFR that the product must fulfill as normalized weights ranging from 0 to 1 as external parameters when executing the transformation. The transformation process uses the relative importance of each NFR and the impact relationships among transformations and quality attributes to select the architectural transformation to be applied. In the automotive example, if the architect selects both the latency and the fault tolerance as being of equal importance (i.e., with a weight of 0.5 for each one) the transformation process will select the *Homogenous Redundancy Pattern (HR)*. The architecture resulting from the application of the HR pattern is shown in Fig. 5. This activity can be performed until all the desired quality attributes for the product are fulfilled.

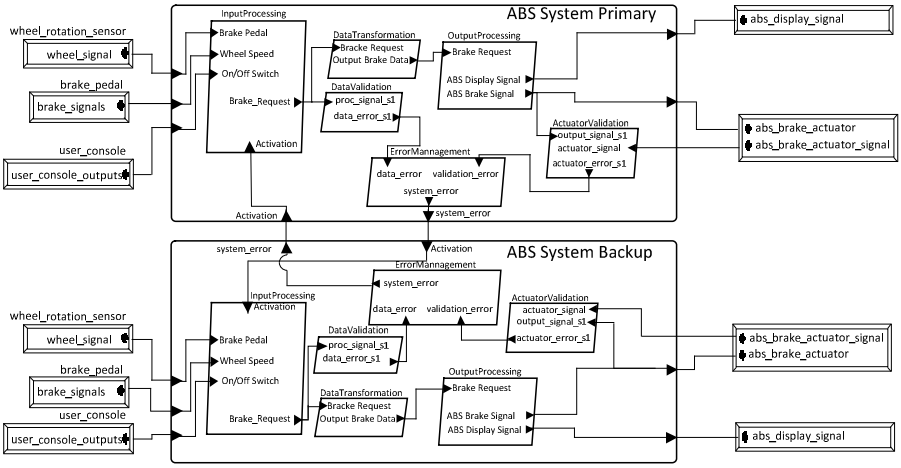

Fig. 5. Product architecture after applying the HR pattern

4 Validation

A controlled experiment was conducted to empirically validate QuaDAI comparing the efficiency, effectiveness and perceived satisfaction of participants using this method against ATAM, a well-known and widely-used software architecture evaluation method [26]. We focus on two activities from the QuaDAI process that occur after deriving the product architecture: *Product Architecture Evaluation* and *Product Transformation*. These activities deal with the evaluation and improvement of product architectures, which are aligned with the main purpose of ATAM.

4.1 Experiment Planning

The controlled experiment was designed by considering the guidelines proposed by Wohlin et al. [34]. According to the Goal-Question Metric (GQM) paradigm [3], the goal of the experiment is to **analyze** the Quality-Driven Architectural Improvement

method (QuaDAI) and ATAM **for the purpose** of comparing them **with respect to** their effectiveness, efficiency, ease of use, usefulness and intention of use in order to obtain software architectures that meet a given set of quality requirements **from the viewpoint** of novice software architecture evaluators.

The context of the experiment is the quality evaluation of two software architectures carried out by novice evaluators. This context is determined by the software architectures to be evaluated, the architecture evaluation methods to be applied and the subject selection.

The *software architectures* to be evaluated are the software architecture of an Antilock Braking System *(ABS System)* from an automotive control system and the software architecture of the *Savi* application (http://goo.gl/1Q49O), a mobile application for emergency notifications. The architecture of the *ABS System,* represented through its component and connector view, was selected as experimental object O1, and the *Savi* architecture, represented through the deployment view, was selected as experimental object O2. We also selected a set of four architectural patterns that can be applied to improve the quality attribute levels of interest in each of the product architectures. The experimental tasks include the evaluation of these quality attributes by means of two software metrics in each experimental object before and after applying the architecture evaluation methods. Thirty-one subjects were selected from a group of fifth-year Computer Science students at the Universitat Politècnica de València who were enrolled on an Advanced Software Engineering course from September 2012 to January 2013, where they acquire knowledge and skills on software architecture evaluation. In particular, they received a training of eight hours on this topic before the experiment took place. The evaluation methods being compared are, on the one hand our proposal described in Section 3 (QuADAI) and on the other, the Architecture Trade-Off Analysis Method (ATAM). ATAM is used to assess the consequences of architectural design decisions in the light of quality attributes [22]. The main goals of ATAM are to elicit and refine the architecture's quality goals; to elicit and refine the architectural design decisions and to evaluate the architectural design decisions in order to determine whether they address the quality attribute requirements satisfactorily. ATAM has been selected for comparison with QuaDAI since i) it is a widely used software architecture evaluation method ii) it is able to deal with multi-attribute analysis [1] and iii) it can be used to evaluate both product line and product architectures at various stages of SPL development (conceptual, before code, during development, or after deployment) [9].

The **independent variable** of interest in the study is the use of each method (ATAM or QuaDAI). There are two **objective dependent variables:** *effectiveness* of the method, which is calculated as a function of the *Euclidean Distances* between the NFR values attained by the architecture being evaluated by the subject and the optimal NFR values that can be attained; and *efficiency*, which is calculated as the ratio between the effectiveness and the total time spent on applying the evaluation method. There are also three **subjective dependent variables:** *perceived ease of use***,** which refers to the degree to which evaluators believe that learning and using a particular method will be effort-free, *perceived usefulness*, which refers to the degree to which evaluators believe that using a specific method will increase their job performance within an organizational context and *intention to use*, the extent to which

a evaluator intends to use a particular method. This last variable represents a perceptual judgment of the method's efficacy – that is, whether it is cost-effective and is commonly used to predict the likelihood of acceptance of a method in practice. These three subjective variables were measured by using a *Likert* scale questionnaire with a set of specific closed questions related to each variable. The aggregated value of each subjective variable was calculated as the mean of the answers to the variable-related questions.

Effectiveness is calculated by applying the formula (1) to normalized *euclidean distances*. The normalization is calculated by applying the formula (2) to the e*uclidean distances* calculated by applying the formula (3) and returns a value ranging from 0 to 1. The normalization is required for avoiding the effects of the scales of the metrics that measure each NFR. The *optimal* function in formulas (1) and (2) returns the optimal values of the NFRs that can be achieved for a given experimental object. The *Max* function returns the maximal distance *D* observed for a given experimental object.

$$Effectiveness(p) = 1 - Norm\big(D(p, optimal(Object))\big) \tag{1}$$

$$Norm\big(D(p, Optimal(Object))\big) = \frac{D(p, Optimal(Object))}{Max(Object)} \tag{2}$$

$$D(p,q) = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2} \tag{3}$$

The **hypotheses** of this experiment are:

- **$H1_0$:** There is no significant difference between the effectiveness of QuaDAI and ATAM / **$H1_a$:** QuaDAI is significantly more effective than ATAM.
- **$H2_0$:** There is no significant difference between the efficiency of QuaDAI and ATAM / **$H2_a$:** QuaDAI is significantly more efficient than ATAM.
- **$H3_0$:** There is no significant difference between the perceived ease of use of evaluators applying QuaDAI and ATAM / **$H3_a$:** QuaDAI is perceived as easier to use than ATAM.
- **$H4_0$:** There is no significant difference between the perceived usefulness of QuaDAI and ATAM / **$H4_a$:** QuaDAI is perceived as more useful than ATAM.
- **$H5_0$:** There is no significant difference between the intention to use of QuaDAI and ATAM / **$H5_a$:** QuaDAI is perceived as more likely to be used than ATAM.

4.2 Experiment Operation and Execution

The experiment was planned as a balanced within-subject design with a confounding effect, signifying that the same subjects executed both methods with both experimental objects in different order. We established four groups (each group applying one method with one object) and the subjects were randomly assigned to each group. Table 1 shows the schedule of the experiment in more detail.

Several documents were designed as instrumentation for the experiment: slides for training session, an explanation of the methods, forms for gathering data, the patterns description, the metrics documentation, and two questionnaires. Excel spread sheets were also designed in order to automate the metrics calculation and the QuaDAI's trade-off among architectural transformations. The instrumentation of this experiment is available at http://www.dsic.upv.es/~jagonzalez/MODELS2013/instrumentation.

A pilot experiment was conducted beforehand to assess the experimental material and to estimate the time required to accomplish the tasks. This took place with four Computer Science PhD students from the Universitat Politecnica de Valencia. The students completed the experimental tasks in less than an hour. This pilot experiment also allowed us to collect information on how to improve the instrumentation.

The experiment was planned to be conducted in three sessions, Table 1 shows the details for each day. On the first day, the subjects were given the complete training on the methods to be applied and also on the tasks to be performed in the execution of the experiment. On the second and third days the subjects were given an overview of the complete training before applying one evaluation method on an experimental object (O1 or O2). We established a slot of 60 minutes without a time limit for each of the methods to be applied.

The experiment took place in a single room, and no interaction between subjects was allowed. The questions that arose during the session were clarified by the same conductors during the experiment.

With regard to the data validation, we verified that one of the subjects had not completed the 2nd session and that it was therefore necessary to eliminate his first exercise. Since we had 30 subjects distributed in four groups, it was necessary to discard two subjects (which were selected randomly) in order to maintain the balanced design, consisting of a total of 28 subjects, seven in each group.

Table 1. Schedule of the controlled experiment

1st session (120 min)	Training on Software Architecture Evaluation using ATAM and QuaDAI			
2nd session (60 + 60 minutes)	Software Architecture Evaluation using ATAM and QuaDAI (short training)			
	QuaDAI in O1	QuaDAI in O2	ATAM in O1	ATAM in O2
	QuaDAI Questionnaire		ATAM Questionnaire	
2nd session (60 + 60 minutes)	Software Architecture Evaluation using ATAM and QuaDAI (short training)			
	ATAM in O2	ATAM in O1	QuaDAI in O2	QuaDAI in O1
	ATAM Questionnaire		QuaDAI Questionnaire	

4.3 Data Analysis

The quantitative analysis was performed by using the SPSS v16 statistical tool using an α=0.05. A summary of the results of the evaluation is shown in Table 2. Mean and standard deviations have also been used as descriptive statistics for the qualitative subjective variables *Perceived Ease of Use (PEOU), Perceived Usefulness (PU)* and *Intention to Use (ITU)*. The five-point *Likert* scale ranging from 1 to 5 adopted for the measurement of the subjective variables has also been considered as an interval scale [9]. The cells highlighted in bold type in Table 2 show the best values for each of the

statistics. These results can be used to interpret that the subjects' best performance was with QuaDAI in almost all the variables.

Table 2. Descriptive results

	Effectiveness		**Efficiency**		**Duration (min)**	
	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.
QuaDAI	**0.68**	0.39	**0.029**	0.018	**25.36**	7.26
ATAM	0.63	0.36	0.020	0.013	31.11	9.15
	Perceived Ease of Use (PEOU)		**Perceived Usefulness (PU)**		**Intention to Use (ITU)**	
	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.
QuaDAI	**3.98**	0.88	**3.80**	0.83	**3.65**	0.84
ATAM	3.50	0.82	3.72	0.73	3.55	0.70

The sample size (<50) indicated that it was necessary to apply the Shapiro-Wilk test to check whether the data was normally distributed so as to select which tests were needed to test the five hypotheses. Table 3 shows the results of the normality test. The variables that are normally distributed for a given architecture evaluation method are shown in bold type.

Table 3. Shapiro-Wilk normality test results

	Effect.	**Effic.**	**PEOU**	**PU**	**ITU**
QuaDAI	0.000	**0.362**	0.014	0.027	0.024
ATAM	0.000	**0.379**	0.027	0.04	**0.894**

The boxplots in Fig. 6 containing the distribution of each dependent variable per subject per method show that QuaDAI was more effective and efficient, and also that it was perceived as being easier to use, more useful and more likely to be used by the subjects than ATAM.

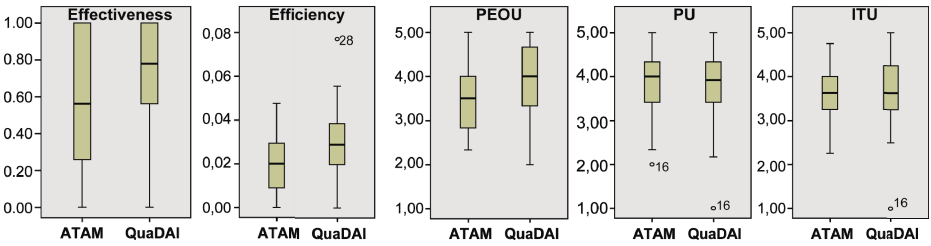

Fig. 6. Boxplots for the various dependent variables

In order to check the statistical significance of these tests we performed the Mann-Withney non-parametric test so as to verify H1, H3, H4, since they are not normally distributed, and H5 and the 1-tailed *t*-test for independent samples to verify H2.The Mann-Whitney test results were 0.906 for *Effectiveness*, 0.030 for *PEOU*, 0.941 for *PU* and 0.767 for *ITU*. The *p*-value obtained from the 1-tailed *t*-test for *Efficiency* was

0.015. These results led us to conclude that the difference in terms of *Efficiency* and *PEOU* is statistically significant, thus allowing us to reject the null hypotheses $H1_0$ and $H3_0$ and accept their respective alternative hypotheses. However, with regard to the *Effectiveness*, *PU* and *ITU*, although the subjects achieved their best results with QuaDAI, we found that the differences were not statistically significant (> 0.05).

4.4 Threats to the Validity

The main threats to the **internal validity** are: learning effect, subjects' experience, information exchange among participants, author's bias, author influence, the order of methods in the training and understandability of the documents. Two experimental objects were used to deal with the learning effect, such as ensuring that each subject applied each method in a different system and considering all the possible combinations of both the method order and the experimental objects. There were no differences on the subjects' experience since none of them had experience in architecture evaluations. The subjects were introduced to the tasks and the problems they would have to solve via their participation in training sessions on both methods. Information exchange was alleviated by using different experimental objects at the same time, and monitoring the subjects while they performed the tasks. Since the experiment was designed to take place in two sessions, the subjects might have been able to exchange information during the time between the sessions, but this was alleviated by asking the participants to return the material at the end of each session. The author's bias in this experiment may have influenced the results since the training sessions were conducted by an author of the method. The author influence was alleviated by not disclosing to the subjects the authorship of the QuaDAI method. The order of methods during the training and experimental sessions could have also influenced the results since it was the same in each session. This issue will be investigated in future replications of this experiment. The understandability of the material was alleviated by clearing up all the misunderstandings that appeared in the pilot experiment and experimental sessions.

The main threat to **external validity** is the representativeness of the results. The representativeness of the results might be affected by the evaluation design and the participant context selected. The evaluation design might have had an impact on the results owing to the kind of architectural models and quality attributes to be evaluated. We selected two different architectures, from two different domains, two different NFRs and four different patterns for each experimental object. The experiment was conducted with students with no experience in architectural evaluations, and who received only limited training on the evaluation methods. However, since they were final year students they can be considered as novice users of architectural evaluation methods, and the next generation of practitioners [24]. The results could thus be considered as representative of novice evaluators.

The main threats to the **construct validity** are the measures used to quantify the dependent variables. Effectiveness was measured using the Euclidean distance which has commonly been used to measure the goodness of a solution with regard to a set of opposed NFRs with different purposes [12] [33]. The subjective variables are based on the Technology Acceptance Method (TAM) [13], a well-known and empirically validated model for the evaluation of information technologies. The reliability of the

questionnaire was tested by applying the Cronbach test. Questions related to PEOU, PU and ITU obtained a Cronbach's alpha of 0.824, 0.870 and 0.831, which is higher than the acceptable minimum (0.70) [27]. The main threat to the **conclusion validity** is the validity of the statistical tests applied. This threat was alleviated by applying a set of commonly accepted tests employed in the empirical SE community [27]. However, more replications are needed in order to confirm these results.

5 Conclusions and Future Work

In this paper, we have presented QuaDAI, a method for the derivation, evaluation and improvement of product architectures. This method relies on a multimodel that represents the different viewpoints of the SPL (functional, quality, variability, and transformation), allowing the representation of the product line architecture, the metrics for its evaluation, and the relationships among architectural transformations and NFRs. The approach has three major benefits: i) it is aimed to automate the derivation and improvement of product architectures; ii) it provides a systematic mechanism for dealing with the cases in which the NFRs of the product under development are not within the range of values permitted by the architectural variability; iii) and finally, it takes advantage of the reuse of the architectural knowledge stored in the multimodel for helping designers to decide which architectural patterns should be applied each time a product architecture needs to be improved. We believe that QuaDAI is useful to guide novice architects in performing evaluations as the multimodel explicitly represents the domain expert's knowledge.

We have also validated our method by means of a controlled experiment in which QuaDAI were compared with a widely-used architecture evaluation method (ATAM). The results show that QuaDAI is more efficient and is perceived to be easier to use than ATAM. However, with regard to the effectiveness, PU and ITU, although QuaDAI achieved better results, we found that the differences were not statistically significant. This may be because the lack of experience of the subjects in architecture evaluation. This issue will be examined in future replications of this study.

As future work, we plan to characterize those cases in which the variability mechanisms are not sufficient to achieve the NFRs for a given product. We also plan to study other mechanisms for introducing the relative importance (weights) for the NFRs. Currently, we are using only numbers but we are aware that they may not capture the full range of real-world impact relationships. We will explore the definition of functions that could express conditions on such numbers. In addition, we are aware that not only architectural patterns can be applied to improve a quality attribute. Our approach may also allow managing other complementary architectural transformations that may be needed.

We also plan to conduct replications of this experiment by considering a larger number of subjects with different subject profiles (e.g., practitioners or students with a higher level of knowledge and skills on architecture evaluation) and different experimental objects in order to improve the representativeness of our results.

Acknowledgements. This research is supported by the MULTIPLE project (MICINN TIN2009-13838) and the ValI+D fellowship program (ACIF/2011/235).

References

1. Ali-Babar, M., Lago, P., Van Deursen, A.: Empirical research in software architecture: opportunities, challenges, and approaches. Empirical Software Engineering 16(5), 539–543 (2011)
2. Ali-Babar, M., Zhu, L., Jeffery, R.: A Framework for Classifying and Comparing Software Architecture Evaluation Methods. In: 15th Australian Software Engineering Conference, Melbourne, Australia, pp. 309–318 (2004)
3. Basili, V.R., Rombach, H.D.: The TAME project: towards improvement-oriented software environments. IEEE Transactions on Software Engineering 14(6), 758–773 (1988)
4. Barkmeyer, E.J., Feeney, A.B., Denno, P., Flater, D.W., Libes, D.E., Steves, M.P., Wallace, E.K.: Concepts for Automating Systems Integration NISTIR 6928. National Institute of Standards and Technology, U.S. Dept. of Commerce (2003)
5. Bosch, J.: Design and Use of Software Architectures. Adopting and Evolving Product-Line Approach. Addison-Wesley, Harlow (2000)
6. Botterweck, G., O'Brien, L., Thiel, S.: Model-driven derivation of product architectures. In: 22th Int. Conf. on Automated Software Engineering, New York, USA, pp. 469–472 (2007)
7. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: Pattern-Oriented software architecture, vol. 1: A System of Patterns. Wiley (1996)
8. Cabello, M.E., Ramos, I., Gómez, A., Limón, R.: Baseline-Oriented Modeling: An MDA Approach Based on Software Product Lines for the Expert Systems Development. In: 1st Asia Conference on Intelligent Information and Database Systems, Vietnam (2009)
9. Carifio, J., Perla, R.J.: Ten Common Misunderstandings, Misconceptions, Persistent Myths and Urban Legends about Likert Scales and Likert Response Formats and their Antidotes. Journal of Social Sciences 3(3), 106–116 (2007)
10. Clements, P., Northrop, L.: Software Product Lines: Practices and Patterns. Addison-Wesley, Boston (2007)
11. Czarnecki, K., Kim, C.H.: Cardinality-based feature modeling and constraints: A progress report. In: Int. Workshop on Software Factories, San Diego-CA (2005)
12. Datorro, J.: Convex Optimization & Euclidean Distance Geometry. Meboo Publishing (2005)
13. Davis, F.D.: Perceived usefulness, perceived ease of use and user acceptance of information technology. MIS Quarterly 13(3), 319–340 (1989)
14. Douglass, B.P.: Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems. Addison-Wesley, Boston (2002)
15. Feiler, P.H., Gluch, D.P., Hudak, J.: The Architecture Analysis & Design Language (AADL): An Introduction. Tech. Report CMU/SEI-2006-TN-011. SEI, Carnegie Mellon University (2006)
16. Gómez, A., Ramos, I.: Cardinality-based feature modeling and model-driven engineering: Fitting them together. In: 4th Int. Workshop on Variability Modeling of Software Intensive Systems, Linz, Austria (2010)
17. Gonzalez-Huerta, J., Insfran, E., Abrahao, S.: A Multimodel for Integrating Quality Assessment in Model-Driven Engineering. In: 8th International Conference on the Quality of Information and Communications Technology (QUATIC 2012), Lisbon, Portugal, September 3-6 (2012)
18. Gonzalez-Huerta, J., Insfran, E., Abrahao, S., McGregor, J.D.: Non-functional Requirements in Model-Driven Software Product Line Engineering. In: 4th Int. Workshop on Non-functional System Properties in Domain Specific Modeling Languages, Insbruck, Austria (2012)

19. Guana, V., Correal, V.: Variability quality evaluation on component-based software product lines. In: 15th Int. Software Product Line Conference, Munich, Germany, vol. 2, pp. 19.1–19.8 (2011)
20. Insfrán, E., Abrahão, S., González-Huerta, J., McGregor, J.D., Ramos, I.: A Multimodeling Approach for Quality-Driven Architecture Derivation. In: 21st Int. Conf. on Information Systems Development (ISD 2012), Prato, Italy (2012)
21. ISO/IEC 25000:2005, Software Engineering. Software product Quality Requirements and Evaluation SQuaRE (2005)
22. Kazman, R., Klein, M., Clements, P.: ATAM: Method for Architecture Evaluation (CMU/SEI-2000-TR-004, ADA382629). Software Engineering Institute, Carnegie Mellon University, Pittsburgh (2000), http://www.sei.cmu.edu/publications/documents/00.reports/00tr004.html
23. Kim, T., Ko, I., Kang, S., Lee, D.: Extending ATAM to assess product line architecture. In: 8th IEEE Int. Conference on Computer and Information Technology, Sydney, Australia, pp. 790–797 (2008)
24. Kitchenham, B.A., Pfleeger, S.L., Hoaglin, D.C., Rosenber, J.: Preliminary Guidelines for Empirical Research in Software Engineering. IEEE Transactions on Software Engineering 28(8) (2002)
25. Kruchten, P.B.: The Rational Unified Process: An Introduction. Addison-Wesley (1999)
26. Martensson, F.: Software Architecture Quality Evaluation. Approaches in an Industrial Context. Ph. D. thesis, Blekinge Institute of Technology, Karlskrona, Sweden (2006)
27. Maxwell, K.: Applied Statistics for Software Managers. Software Quality Institute Series. Prentice-Hall (2002)
28. Olumofin, F.G., Mišic, V.B.: A holistic architecture assessment method for software product lines. Information and Software Technology 49, 309–323 (2007)
29. Perovich, D., Rossel, P.O., Bastarrica, M.C.: Feature model to product architectures: Applying MDE to Software Product Lines. In: IEEE/IFIP & European Conference on Software Architecture, Helsinki, Findland, pp. 201–210 (2009)
30. Robertson, S., Robertson, J.: Mastering the requirements process. ACM Press, New York (1999)
31. Roos-Frantz, F., Benavides, D., Ruiz-Cortés, A., Heuer, A., Lauenroth, K.: Quality-aware analysis in product line engineering with the orthogonal variability model. Software Quality Journal (2011), doi:10.1007/s11219-011-9156-5
32. Saaty, T.L.: The Analytical Hierarchical Process. McGraw- Hill, New York (1990)
33. Taher, L., Khatib, H.E., Basha, R.: A framework and QoS matchmaking algorithm for dynamic web services selection. In: 2nd Int. Conference on Innovations in Information Technology, Dubai, UAE (2005)
34. Wohlin, C., Runeson, P., Host, M., Ohlsson, M.C., Regnell, B., Weslen, A.: Experimentation in Software Engineering - An Introduction. Kluwer (2000)