A Survey on Software Architecture Analysis Methods

Liliana Dobrica and Eila Niemelä, *Member*, *IEEE Computer Society*

Abstract—The purpose of the architecture evaluation of a software system is to analyze the architecture to identify potential risks and to verify that the quality requirements have been addressed in the design. This survey shows the state of the research at this moment, in this domain, by presenting and discussing eight of the most representative architecture analysis methods. The selection of the studied methods tries to cover as many particular views of objective reflections as possible to be derived from the general goal. The role of the discussion is to offer guidelines related to the use of the most suitable method for an architecture assessment process. We will concentrate on discovering similarities and differences between these eight available methods by making classifications, comparisons and appropriateness studies.

Index Terms—Software architecture, analysis techniques and methods, quality attributes, scenarios.

1 Introduction

One of the major issues in software systems development today is quality. The idea of predicting the quality of a software product from a higher-level design description is not a new one. In 1972, Parnas [44] described the use of modularization and information hiding as a means of high-level system decomposition to improve flexibility and comprehensibility. In 1974, Stevens et al. [52] introduced the notions of module coupling and cohesion to evaluate alternatives for program decomposition. During recent years, the notion of software architecture (SA) has emerged as the appropriate level for dealing with software quality. This is because the scientific and industrial communities have recognized that SA sets the boundaries for the software qualities of the resulting system [7].

Recent efforts towards the systematization of the implications of using design patterns and architectural styles contribute, frequently in an informal way, to the guarantee of the quality of a design [16], [19]. It is recognized that it is not possible to measure the quality attributes of the final system based on SA design [12]. This would imply that the detailed design and implementation represent a strict projection of the architecture. The aim of analyzing the architecture of a software system is to predict the quality of a system before it has been built and not to establish precise estimates but the principal effects of an architecture [27]. The purpose of the evaluation is to analyze the SA to identify potential risks and verify that the quality requirements have been addressed in the design [38].

- *L. Dobrica is with the Faculty of Automatic Control and Computers, University Politehnica of Bucharest, Spl. Independentei 313, Sect. 6, Bucharest, 77206 Romania. E-mail: liliana@ciid.pub.ro.*
- *E. Niemelä is with the Software Architectures Group, Embedded Software, VTT Electronics, P.O. Box 1100 FIN-90571 Oulu, Finland. E-mail: eila.niemela@vtt.fi.*

Manuscript received 3 July 2000; revised 22 Mar. 2001; accepted 9 July 2001. Recommended for acceptance by D. Rosenblum.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number 112394.

More formal efforts are concentrated on ensuring that the quality is addressed at the architectural level. Different communities of the software metrics, scenario-based, and attribute model-based analysts have developed their own techniques. The software metrics community has used module coupling and cohesion notions to define predictive measures of software quality [15]. Other methods include a more abstract evaluation of how the SA fulfills the domain functionality and other nonfunctional qualities [27]. Instead of presenting metrics for predictive evaluation, they exemplify the argument for performing a more qualitative or quantitative evaluation. Methods based on scenarios could be considered mature enough since they have been applied and validated over the past several years, but the development of attribute model-based architecture evaluation methods is still ongoing.

Future work is needed to develop systematic ways of bridging quality requirements of software systems with their architecture. The open problem is how to take better advantage of software architectural concepts to analyze software systems for quality attributes in a systematic and repetitive way. Being a new research domain, most of the structural methods for assessing the quality of SAs have been presented in conference and journal papers. Although refinement and experiments for validating some of the methods are ongoing, they deserve our attention because they contribute to the development of what is still an immature research area. Therefore, we decided to study such methods in order to cover as many particular points of view of objective reflections as possible to be derived from the general goal. SA is considered the first product in an architecture-based development process and, from this point of view, the analysis at this level should reveal requirement conflicts and incomplete design descriptions from a particular stakeholder's perspective. The analysis could be associated with the design of an iterative improvement of the architecture of a green software system or with the reengineering of an existent one. Prediction methods of a single quality attribute are meant to minimize

risks only from that attribute perspective at a fine level. This might not be sufficient if the quality of a system is represented by a variety of attributes that interact with each other and a balance between them should be established.

The discussed methods include the scenario-based architecture analysis method (SAAM) [26] and its three particular cases of extensions, one founded on complex scenarios (SAAMCS) [35], and two extensions for reusability, ESAAMI [43] and SAAMER [41], the architecture trade-off analysis method (ATAM) [29], scenario-based architecture reengineering (SBAR) [8], architecture level prediction of software maintenance (ALPSM) [10], and a software architecture evaluation model (SAEM) [18].

This survey puts all these developments in the same perspective by reviewing the state of the software architecture analysis methods. The beginning of this study is dedicated to the definitions of the terminology that is frequently used in the context of the methods. Based on these general elements and others related to methodology characterization, we define a conceptual framework for presentation and comparison of the analysis methods while trying to look for 1) their progress towards refinement over time, 2) their main contributions, and 3) advantages obtained by using them. The discussions surrounding the selected methods focus on 1) discovering differences and similarities and 2) making classifications, comparisons and appropriateness studies. Finally, we will draw conclusions from the real level of the current research as well as the future work in this domain defined by the presented methods.

2 Definitions of the Main Terminology

2.1 Quality Attributes and the Quality Model

A quality attribute is a nonfunctional characteristic of a component or a system. A software quality is defined in IEEE 1061 [22] and it represents the degree to which software possesses a desired combination of attributes. Another standard, ISO/IEC Draft 9126-1 [23], defines a software quality model. According to this, there are six categories of characteristics (functionality, reliability, usability, efficiency, maintainability, and portability), which are divided into subcharacteristics. These are defined by means of externally observable features for each software system. In order to ensure its general application, the standard does not determine which these attributes are nor how they can be related to the subcharacteristics.

An investigation into the literature has shown that a large number of definitions of quality attributes exist that are related to similar abilities of a software system. For example, maintainability, flexibility, and modifiability definitions are described as follows:

Maintainability is a set of attributes that have a bearing on the effort needed to make specified modifications [23]. Modifications may include corrections, improvements or adaptations of software to changes in environment, and in requirements and functional specification.

Modifiability is the ability to make changes quickly and cost-effectively [7]. Modifications to a system can be categorized as *extensibility* (the ability to acquire new features), *deleting unwanted capabilities* (to simplify the functionality of an existing application), *portability* (adapting to new operating environments), or *restructuring* (rationalizing system services, modularizing, creating reusable components).

Flexibility is the ease with which a system or component can be modified for use in applications or environments other than those for which it was specifically designed [21].

Although different in wording, the definitions are almost identical in their semantics. The limitation of these definitions with respect to the purpose of analyzing SAs is that their scope is too broad. The scope has to be narrowed, based on the relevant context.

2.2 Software Architecture Definition and Description

Definition. The software architecture of a system is defined as "the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them" [7]. This definition focuses only on the internal aspects of a system and most of the analysis methods are based on it. Another brief definition given by Garlan and Perry [45], [49] establishes SA as "the structure of components in a program or system, their interrelationships, and the principles and guides that control the design and evolution in time," This process-centered definition is used by SAEM because it takes into account the presence of principles and guidelines in the architecture description. For an analysis of flexibility, the external environment is just as important as an internal entity of a system [37]. The SA definition should consist of two parts, namely of a macroarchitecture which focuses on the environment of the system, and a microarchitecture which covers the internal structure of a system.

Description. The research in the SA description addresses the different perspectives one could have of the architecture. Each perspective is described as a view. Although there is still no general agreement about which views are the most useful, the reason behind multiple views is always the same: *Separating different aspects into separate views help people manage complexity.* Several models have been proposed that introduce a number of views that should be described in the SA [33]. The view models share the fact that they address a static structure, a dynamic aspect, a physical layout, and the development of the system. Bass et al. [7] introduce the concept of architecture structures as being synonymous to view. In general, it is the responsibility of the architect to decide which view to use to describe the SA.

From the point of view of quality analysis at the architectural level, the possible representations could be very relevant in quality prediction and effort estimation (Fig. 1). An evaluation method may need structures, which are concerned with the decomposition of the functionality that the products need to support, the realization in a detailed design of the conceptual abstractions from which the system is built, logical concurrency, hardware, files, and directories. Components and relations of each structure are representative. For instance, logical concurrency structure

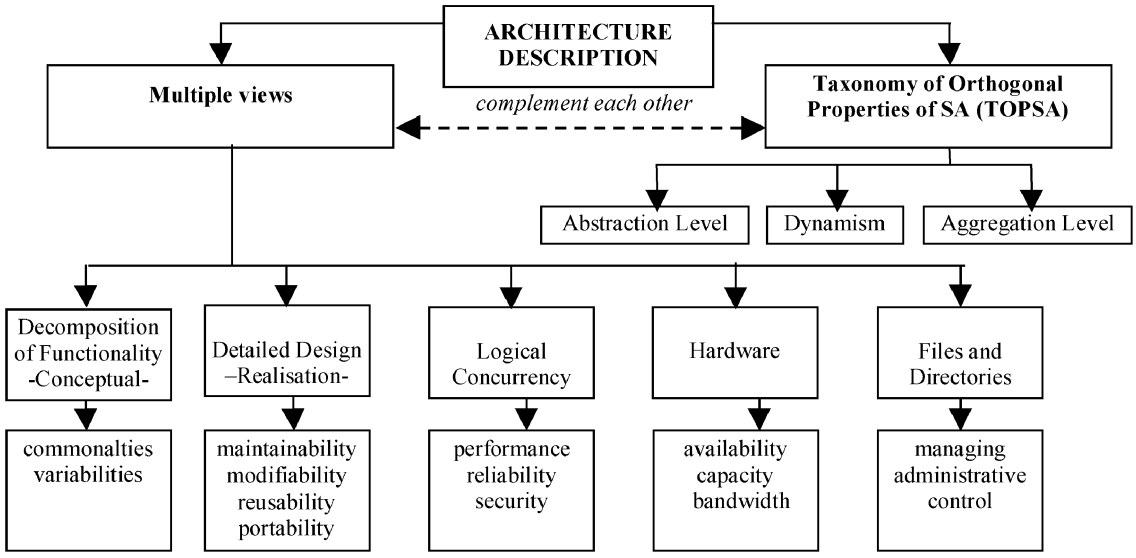

Fig. 1. Architecture description and the relevance to analysis of quality attributes.

contains units that are refined to processes and threads. Its relations include *synchronizes-with*, *is-higher-priority-than*, *sends-data-to*, *can't-run-without*, etc. Properties relevant to this structure include priority, preemptibility, and execution time.

A taxonomy of formally-defined orthogonal properties of SAs (TOPSA) that extends the first SA definition is given in [14]. The TOPSA space has three dimensions: *abstraction level* (conceptual, realization), *dynamism* (static, dynamic), and *aggregation level* that can facilitate discussions regarding SA during development and evolution. The TOPSA and the architecture representation based on multiple views complement each other (Fig. 1). Different views offer valuable examples for abstraction, dynamism, and aggregation dimensions. An analysis method can exploit these relationships in the form of a defined set of rules, which states which view in the TOPSA space is the most appropriate for a given quality attribute.

Descriptions of component types and their topology, of patterns of data and control interactions among the components, and the benefits and drawbacks of using them are also documented in architectural styles [49], [16] and design patterns [19]. Compositions of patterns that might be used in a SA are evaluated in various quality terms in [32].

2.3 Evaluation Techniques at the Architecture Level

Two basic classes of evaluation techniques, questioning and measuring, available at the architecture level are defined in two important research reports [1], [7]. *Questioning techniques* generate qualitative questions to be asked of an architecture and they can be applied for any given quality. This class includes scenarios, questionnaires, and checklists. *Measuring techniques* suggest quantitative measurements to be made on an architecture. They are used to answer specific questions and they address specific software qualities and, therefore, they are not as broadly applicable as questioning techniques. This class includes metrics, simulations, prototypes and experiences. Generality, level-of-detail, phase, and what is evaluated represent a four-dimensional framework of comparison of these techniques [7]. Regarding generality, the techniques could be general (questionnaire), domain-specific (checklists, prototype), or system-specific (scenarios). The level of detail (coarse-grained, medium, or fine) indicates how much information about the architecture is required to perform the evaluation. There are three phases of interest to architecture evaluation: early-, middle-, and postdeployment. These phases occur after the initial high-level architectural decisions (questionnaire, prototype), at any point after some elaboration of the architecture design (scenarios, checklists), and after the system has been completely designed, implemented, and deployed.

In terms of quantitative and qualitative aspects, both classes of techniques are needed for evaluating architectures. Various analyzing models expressed in formal methods are included in quantitative techniques. Qualitative techniques illustrate SA evaluations using scenarios. A description of the changes that are needed for each scenario represents a qualitative method of evaluation. From this perspective, scenarios are necessary but not sufficient to predict and control quality attributes and they have to be supplemented with other evaluation techniques and, particularly, quantitative interpretations. For example, including questions about quality indicators in the scenarios enriches the architecture evaluation. Quantitative interpretations of scenario evaluations could be ranking between the effects of scenarios (i.e., a five level scale $(++, +, +/\ ,\ ,\ )$) or an absolute statement, which estimates the size of modifications or different metrics, such as lines of code, function points, or object points.

Most of the considered architecture analysis methods use *scenarios*. The existing practices with scenarios are systematized in [30]. Scenarios are a good way of synthesizing individual interpretations of a software quality into a common view [34]. This view is more concrete than the general definition of software quality [21] and it also incorporates the specifics of a system to be developed (i.e., it is more context-sensitive [3]).

TABLE 1
Framework Elements for Characterization and Comparison of Analysis Methods

Framework elements	*Description*
Specific method's goal	What is the specific goal of the method?
Evaluation techniques	Which of the evaluation techniques are included in the method?
Quality attributes	On what quality attributes and what number of quality attributes is the method considered to have specific assessment?
SA description	What architectural views are the foci of the analysis methods? Which is the development phase of the SA?
Stakeholders' involvement	Which stakeholders participate in the evaluation process?
Method's activities	In what order and in what way are the evaluation techniques used to accomplish the method's specific goal? What does the result of the method represent?
Reusability of an existent knowledge base	Is any knowledge base considered? How is it structured for effective reusability?
Method validation	Has the method been validated in practical industrial cases?

Scenarios are a postulated set of uses or modifications of the system. The modifications could be *a change to how one or more components perform an assigned activity, the addition of a component to perform some activity, the addition of a connection between existing components, or a combination of these factors.* In creating and organizing scenarios, it is important that all roles relevant to that system are considered since design decisions may be made to accommodate any of these roles. Different roles represent stakeholders related to a system. Such stakeholders may be *the end user,* who is responsible for executing the software; *the system administrator,* who is responsible for managing the data repositories used by the system; *the developer,* who is responsible for modifying the runtime functions of the system and the organization responsible for approving new requirements.

2.4 A Framework for Characterisation and Comparison of Analysis Methods

A framework for presentation and comparison of the SA analysis methods is described and motivated by the main concepts stated above and methodologies' characterizations (Table 1). Methods include a predefined and organized collection of techniques. However, in addition to a set of techniques, a method should include a set of rules that establishes how to conduct an activity which has a precise goal regarding the result to be achieved. The rules state *by whom, in what order, and in what way the techniques* are used to accomplish the method objective.

3 Overview of Analysis Methods

3.1 Scenario-Based Architecture Analysis Method (SAAM)

SAAM appeared in 1993, corresponding with the trend for a better understanding of general architectural concepts, as a foundation for proof that a software system meets more than just functional requirements [26], [27]. Thus, in the early stage of a system's development, the correction of architectural mistakes detected by the analysis is still possible without causing excessively high costs. The main method's activities are presented in an article where different user interface architectures are assessed with respect to modifiability [28].

Specific goals. A SAAM's goal is to verify basic architectural assumptions and principles against the documents describing the desired properties of an application. Additionally, the analysis offers a contribution to assess the risks inherent to the architecture. SAAM guides the inspection of the architecture focusing on potential trouble spots, such as requirement conflicts or incomplete design specification from a particular stakeholder's perspective. The capability of SAAM to evaluate the suitability of architecture with respect to the desired properties of a specific system can also be used to compare different architectures.

The evaluation technique. Scenarios represent the foundation for illuminating the properties of SA. They illustrate the kinds of activities that the system must support and the kinds of anticipated changes that will be made to the system. During the analysis, it is determined whether a scenario requires modifications to the architecture. Scenarios that require no modifications are called *direct* and those that require modifications are called *indirect*.

The quality attributes. The fundamental characteristic of this method is the concretization of any quality attribute in the form of scenarios. However, it is considered that modifiability is still the quality attribute analyzed by SAAM.

The stakeholders' involvement. SAAM harmonizes various interests of the stakeholder groups, thus setting up a common understanding of the SA as a base for later decisions.

SA description. The method is applied to a final version of the SA but prior to the detailed design. The description of the SA should be in a form that is easily understandable by all stakeholders. Functionality, structure, and allocation are the three perspectives defined for describing SAs. Functionality is what the system does. A small and simple lexicon is used for describing structures for a common level of understanding and comparing different architectures. SA should be presented in a static representation (system

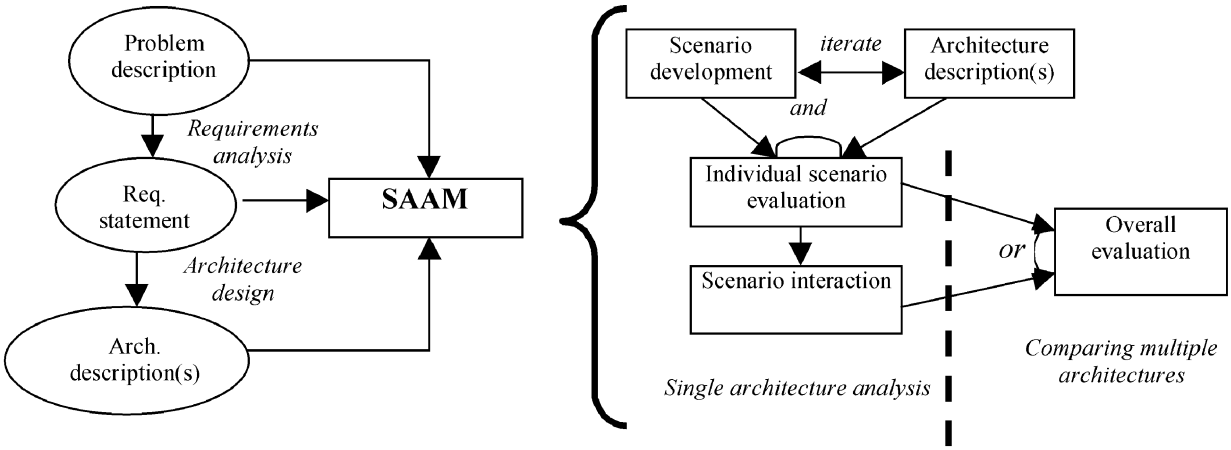

Fig. 2. SAAM inputs and activities.

computation, data components, data and control connections) and a dynamic representation of how the system behaves over time. The allocation of function to structure identifies how the domain functionality is realized in the software structure. The components could be described either as modules in the sense of Parnas [44] or as cooperating sequential processes.

Method's activities. The main inputs of SAAM are problem description, requirements statement and architecture description(s). Fig. 2 presents the inputs associated to the activities of SAAM carried out either for a single architecture or for comparison of multiple ones.

In the case of a single SA analysis, the activities are scenario development, SA description, individual scenario evaluation and scenario interaction. In the scenario development, SAAM requires the presence of all stakeholders that identify possible scenarios as described above. SAAM considers the set of scenarios to be complete when the addition of a new scenario no longer disturbs the design. The second activity, SA description, is recommended to be carried out in parallel with the first activity in an iterative mode. The final version of SA description together with the scenarios serves as the input for the subsequent activities of the method.

SAAM evaluates a scenario by investigating which architectural elements are affected by that scenario. Table 2 is an example of scenario evaluation for an architecture that contains components called A, B, C, D, and E. For a single architecture analysis, the purpose is to determine which scenarios interact, i.e., which ones affect the same component. The cost of the modifications associated with each indirect scenario is estimated by listing the components and the connectors that are affected and then counting the number of changes. If the analysis is performed with the intention of choosing among several architectural alternatives, the results of the candidates can be compared in a final SAAM activity. To this end, scenarios and the scenario interactions are weighted in terms of their relative importance. This weighting is then used to determine an *overall evaluation* of the candidate architectures.

Results interpretation. High interaction of unrelated scenarios could indicate a poor separation of functionality. The amount of scenario interaction is related to metrics such as structural complexity, coupling, and cohesion and, therefore, is correlated with the number of defects in the final product. SAAM cannot give precise measures or metrics of fitness. The result is a set of small metrics that permits a comparison per scenario-basis of competing SA.

Reusability of the existing knowledge base. SAAM does not consider this issue.

Method validation. SAAM is a mature method, validated in various case studies. An enumeration of the case studies includes global information systems, air traffic control, WRCS (revision control system), user interface development environments, Internet information systems, keyword in context (KWIC) systems, and embedded audio systems.

TABLE 2
Scenario Evaluation

Stakeholder	*Scenario*	*Scenario Description*	*Direct/ Indirect*	*Architecture Changes*
User	U1	….	Indirect	Modifications to A and B components
	U2	….	Direct	-
Maintainer	M1	…	Indirect	Modifications to A, B, C, D and E components
	M2	….	…	…
Administrator	A1	…	…	…
	A2	…	…	…

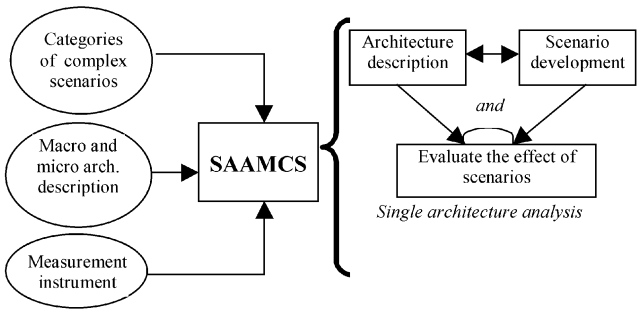

Fig. 3. Inputs and activities of SAAMCS.

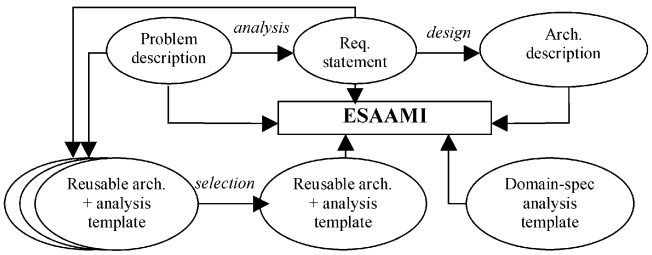

Fig. 4. ESAAMI inputs [43].

3.2 SAAM Founded on Complex Scenarios (SAAMCS)

SAAMCS considers that the complexity of scenarios is the most important factor for risk assessment [35]. SAAMCS contributions for extending SAAM are, on one hand, directed to the way of looking for the scenarios and, on the other, to where their impact is evaluated.

Specific goal. Risk assessment represents the only goal of SAAMCS.

The included evaluation technique. SAAMCS is looking for scenarios that are possibly complex to realize. Based on the initiator of the scenario, SA description, and version conflicts, a list of classes of scenarios that are complicated to implement is provided.

The quality attributes. Flexibility represents the quality attribute analyzed by SAAMCS.

The stakeholders' involvement. The method appreciates stakeholders' involvement and identifies the important role of the initiator of a scenario. The initiator is the organizational unit that has most interest in the implementation of that scenario.

SA description. SAAMCS is applied to the final version of the architecture, which is described in sufficient detail. In this method, the idea of the systems within a domain not being isolated but instead integrated within an environment is advanced. As a result, the description of the SA is divided into macroarchitecture and microarchitecture.

Method's activities. Fig. 3 describes the inputs and activities of SAAMCS. In the scenario development, a two-dimensional framework diagram (five categories of complex scenarios, four sources of changes) that may help to discover complicated scenarios is defined. Sources of changes are functional requirements, quality requirements, external components, and the technical environment. Categories of complex scenarios are adaptations to the system with external effects, to the environment with effects to the system, to the macroarchitecture and to the microarchitecture, and the introduction of version conflicts.

Regarding the scenario impact evaluation, SAAMCS introduces and uses a measurement instrument to express the effect of scenarios. The defined instrument includes factors that influence the complexity of scenarios. Three different factors are identified: *four levels of impact of the scenario* (no impact 1), affects one component 2), affects several components 3), affects SA 4)), *the number of owners involved in the information system* and *four levels regarding the presence of version conflicts* (no problem with different versions 1), the presence is undesirable but not prohibitive 2), creates complications related to configuration management 3), creates conflicts 4)). The results are expressed in a table similar to Table 3.

Reusability of the existing knowledge base. SAAMCS does not consider this issue.

Method validation. SAAMCS has been validated for business information systems.

3.3 Extending SAAM by Integration in the Domain (ESAAMI)

Specific goal. The SAAM applied in an architectural-centric development process considers only the problem description, requirements statement and architecture description.

ESAAMI is a combination of analytical and reuse concepts and is achieved by integrating the SAAM in the domain-specific and reuse-based development process [43]. The degree of reuse is improved by concentrating on the domain. ESAAMI is similar to SAAM with regards to the evaluation technique, the quality attributes, the stakeholder's involvement, and SA description. However, an improvement is seen in the reuse of domain knowledge defined by SAs and analysis templates. Fig. 4 describes the main inputs of ESAAMI and the relationship between them. A reusable SA is packaged with a tailored analysis template focused on the distinctive characteristics of the architecture. All these packages represent inputs for the selection process

TABLE 3
Result of Scenario Evaluation in SAAMCS

Change scenario	*Initiator*	*Macro architecture level*			*Micro architecture level*		
		Impact level	*Multiple owners*	*Versions conflicts*	*Impact level*	*Multiple owners*	*Versions conflicts*
S1	User	1	+/-	4	2	+/-	4
…	…	…	…	…	…	…	…

of a reusable architecture. The selected SA is a starting point for the architecture design, being adapted and refined to meet the new system properties.

SA description. A reusable SA to be deployed in the new system is selected in the first step of ESAAMI. It has to be ensured that SA provides an adequate basis for the system to meet its requirements. Three factors influence the reusability of an architecture. The author identifies a common basis for a variety of systems in a domain, sufficient flexibility to cope with variation among systems, and the documentation of properties to make SA available for selection.

Reusability of the existing knowledge base. ESAAMI proposes packages of analysis templates which represent the essential features of the domain. An analysis template is formulated on an abstraction level defined by the commonalities of the systems in the domain and without referring to system-specific architectural elements. Analysis templates collect reusable products that can be deployed in the various steps of the method. These products are proto-scenarios, evaluation protocols, proto-evaluations, and architectural hints and weights. Protoscenarios are generic descriptions of reuse situations or interactions with the system. These are intended for use in the scenario development phase of subsequent architecture analyses after a selection and refinement process. The other products are used in the scenario evaluation phase and are identified in the protocols of the earlier evaluations in different projects, examples of descriptions of how the scenario can be performed using a set of abstract architecture elements, and hints associated to each scenario indicating which structures would make the scenario convenient to handle. Weights, established in old projects in the domain, can make the results of the analysis comparable.

Method's activities are similar to SAAM, but they consider the existence of a reusable knowledge base. The results of the current analysis are part of the newly built system.

Method validation. The method is still in the improvement process.

3.4 Software Architecture Analysis Method for Evolution and Reusability (SAAMER)

Specific goal. From the point of view of two particular quality attributes, evolution, and reusability, SAAM is extended in SAAMER [41]. SAAMER better suggests how a system could support each of the quality objectives or the risk levels for evolution or how to reuse it.

The evaluation technique. Scenarios are the main drivers for evaluating various areas of SA. They describe an important functionality that the system must support, or recognize where the system may need to be changed over time. Scenarios are developed based on the stakeholders' and architectural objectives and considering the fundamental uses of the system. Scenarios and the structural view are effective in identifying components that need to be modified, or are useful for preventive and adaptive maintenance activities.

The quality attributes. Evolution and reusability are considered. Evolution integrates new quality objectives (maintainability and modifiability) obtained from domain experts.

Stakeholders' involvement is similar to SAAM. Additionally, two kinds of sources of information, the required changes and domain experts' experiences, are considered.

SA description. SAAMER considers the following architectural views as critical: *static, map, dynamic, and resource*. The static view integrates and extends SAAM to address the classification and generalization of a system's components and functions and the connections between components. These extensions facilitate the estimation of cost or effort required for changes to be made. The dynamic view is appropriate for the evaluation of the behavior aspect, to validate the control and communication to be handled in an expected manner. The mapping between components and functions could reveal the cohesion and coupling aspects of a system.

Method's activities. SAAMER provides a framework of activities that are useful for the analysis process. This framework consists of four activities: gathering information about stakeholders [13], SA, quality, and scenarios; modeling usable artifacts; analysis; evaluation. The last two activities are similar to SAAM. However, in the scenario development phase of SAAMER, a practical answer to the question regarding when to stop generating scenarios is given. Two techniques are applied here. First, scenario generation is closely tied to various types of objectives: stakeholder, architecture, and quality. Based on the objectives and domain experts' knowledge, the scenarios are identified and clustered to make sure that each objective is well covered. The second technique applied to validate the balance of scenarios with respect to the objective is Quality Function Deployment (QFD) [13], [17]. From stakeholder and architectural objectives to quality attributes, a cascade of matrices is generated to show the relational strengths. Finally, quality attributes are translated to scenarios to reveal the coverage of each one. An imbalance factor is then calculated for each quality attribute by dividing the coverage by the priority of the quality. If the factor is less than 1, more scenarios should be developed to address the attribute in accord with the stakeholder, SA, and quality importance.

Result interpretation. Analysis of scenario interaction is considered a critical step that provides the result of the analysis. A high degree of interaction may indicate that a component is poorly isolated. Still, an SA view may show that this is just the nature of a particular pattern.

Reusability of the existing knowledge base. SAAMER does not consider this issue.

The method has been applied to a telecommunication software system.

3.5 The Architecture Trade-Off Analysis Method (ATAM)

ATAM has grown out of the work on architectural analysis of individual quality attributes: SAAM for analyzes of modifiability, performance, availability, and security. ATAM was considered a spiral model of designs in 1998 [29] and in May 1999 [25] a spiral model of analysis and design, which explains its recent evolution and progress.

Specific goals. The objective of ATAM is to provide a principle way of understanding an SA's capability with respect to multiple competing quality attributes [5]. ATAM recognizes the necessity of a trade-off among multiple software quality attributes when the SA of a system is specified and before the system is developed.

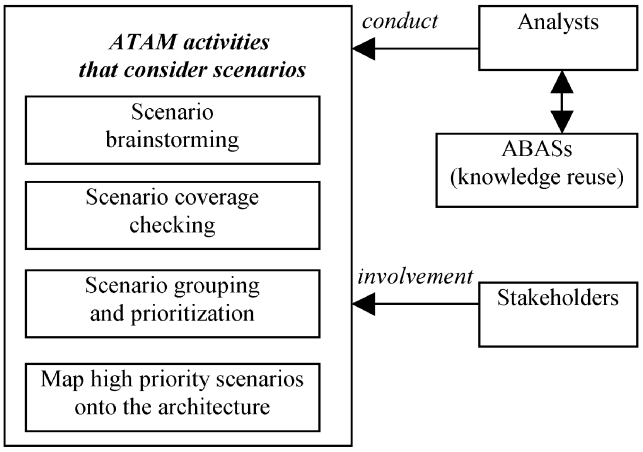

Fig 5. ATAM activities that consider scenarios.

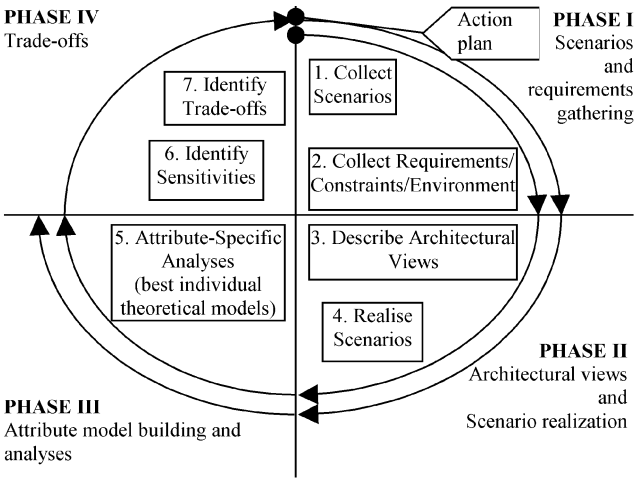

Fig. 6. ATAM phases [29].

The quality attributes. Multiple competing quality attributes are analyzed by ATAM. For the beginning modifiability, security, performance and availability have been considered.

Stakeholders' involvement. ATAM requires all the stakeholders' involvement in the activities related to scenarios and requirements gathering. An SA designer can also be involved.

SA description. The space of architecture is constrained by legacy systems, interoperability, and failures of the previous projects. SA is described on the basis of five foundational structures, which are derived from Kruchten's "4+1 views" [33] (his logical view is divided into function and code structures). These structures plus the appropriate mappings between them can describe an architecture completely. Also, ATAM requires several different views: *a dynamic view*, showing how systems communicate; a *system view*, showing how software was allocated to hardware; and a *source view*, showing how components and systems were composed of objects. SA description is annotated with *a set of message sequences charts* showing run-time interactions and scenarios. ATAM is applied during SA design or on the final version of the SA by an external team of analysts.

The included evaluation techniques. ATAM can be considered a framework for different evaluation techniques depending on the quality attributes. It integrates the best individual theoretical model of each considered attribute in an efficient and practical way [20], [42], [52].

Another evaluation technique is scenario. Three types of scenario probes the system from different architectural views. These are: *use cases*, which involve typical uses of the system and are exploited for the information elicitation; *growth scenarios*, which cover anticipated changes; and *exploratory scenarios*, which cover extreme changes that are expected to "stress" a system. There is a triple scenario role in this method. This technique helps to put vague and unquantified requirements and constraints in concrete terms. Also, scenarios facilitate communication between stakeholders because they force them to agree on their perception of the requirements. Finally, scenarios explore the space defined by an attribute model by helping to put the model parameters that are not part of the SA into concrete terms.

ATAM also considers *qualitative analysis heuristics*, that are derived from an attribute-based architecture style (ABAS) [32] and are meant to be coarse-grain versions of the kind of analysis that is performed when a precise analytic model of a quality attribute is built. An existent taxonomy of each attribute is another base for ATAM. The taxonomies help to ensure attribute coverage and offer a rationale for asking *elicitation questions*. ATAM also uses *screening questions*, which guide or focus the elicitation on more "influential" places of the SA. These serve to limit the portion of the architecture under scrutiny. Asking these questions is more practical than building attribute quantitative models at a moment. They capture the essence of the typical problems that are discovered by a more rigorous and formal analysis.

Method's activities. The method is divided into four main areas of activity, or phases [29]. These are the gathering of scenarios and requirements, architectural views and scenario realization, attribute model building and analysis, and trade-offs. Fig. 5 details activities that include scenarios and Fig. 6 describes the steps associated with each phase and possible iterations for SA design and analysis improvement.

Attribute experts independently create and analyze their models and then they exchange information (clarifying and creating new requirements). The attribute analyses are interdependent because each attribute has implications on others. The attribute interactions are discovered in two ways: using sensitivity analysis to find trade-off points and by examining the assumptions. Recognized from a knowledge base, unbounded *sensitivity points* are informally referred properties that have not yet been bound to the architecture. A *sensitivity point* is a property of one or more components (and/or component relationship) that is critical for achieving a particular quality. In practice, therefore, changes to the architecture parameters significantly affect the modeled values. This can be obtained by using the stimuli and architectural parameters branches of attributes taxonomies [4], [5]. Trade-off points are architectural elements that multiple elements are sensitive to. A *trade-off point* is a property that affects more than one attribute and is a sensitivity point for at least one attribute.

During the architecture design, ATAM provides an iterative improvement. In addition to the requirements typically derived from scenarios that are generated through interviews with the stakeholders, there are assumptions regarding behavior patterns and the execution environment. Because attributes "trade off" against each other, each assumption is subject to inspection, validation, and questioning as a result of ATAM. When all these actions have been completed, the results of the analysis are compared to the requirements. If the system-predicted behavior comes adequately close to its requirements, the designers can proceed to a more detailed level of the design or implementation. In the event of the analysis revealing a problem, an *action plan* for changing the SA, the models or the requirements is developed. This leads to another iteration of the method. ATAM does not require that all attributes be analyzed in parallel, thus allowing the designer to focus on primary attributes, and then introduce others later on. This leads to cost benefits since what may be costly analyses for secondary attributes need not be applied to architecture that was unsuitable for the primary attributes.

Reusability of a domain knowledge base is maintained in ABASs. ABAS helps to move from the notion of architectural styles toward the ability to reason based on quality attribute-specific models. The goals of having a collection of ABASs are to make architectural design more routine-like and more predictable, to have a standard set of attribute-based analysis questions, and to tighten the link between design and analysis [32].

The method has been applied to several software systems but is still under research.

3.6 Scenario-Based Architecture Reengineering (SBAR)

The contribution of this method is not only in the architecture design but also in the scenario-based evaluation of the software qualities of a detailed architecture of a system [8].

Specific goal. SBAR estimates the potential of the designed architecture to reach the software quality requirements.

The included evaluation techniques. Four different techniques for assessing quality attributes are identified: scenarios, simulation, mathematical modeling, and experience-based reasoning. For each quality attribute, the suitable technique is selected. *Scenarios* are recommended for the quality attributes of the development, such as maintainability and reusability, which are exemplified in the paper [8]. The selected scenarios concretize the actual meaning of the attribute (i.e., scenarios that capture typical changes in requirements may specify the maintainability). The performance of the architecture in the context defined by each individual scenario for a quality attribute is assessed by the analysis. *Simulation* completes the scenario-based approach, being useful for evaluating operational software qualities such as performance or fault-tolerance. *Mathematical models* allow a static evaluation of architectural design models and are an alternative to simulation since both approaches are primarily suitable for assessing operational software qualities. To evaluate operational software qualities, the existent mathematical models developed by various research communities for high performance-computing [50], reliability [48], and real-time systems [39] could be used. *Experience-based reasoning* is founded on experience and logical reasoning based on that experience. This technique is different from the others because is less explicit and more based on subjective factors such as intuition and experience and it makes use of the tacit knowledge of the people.

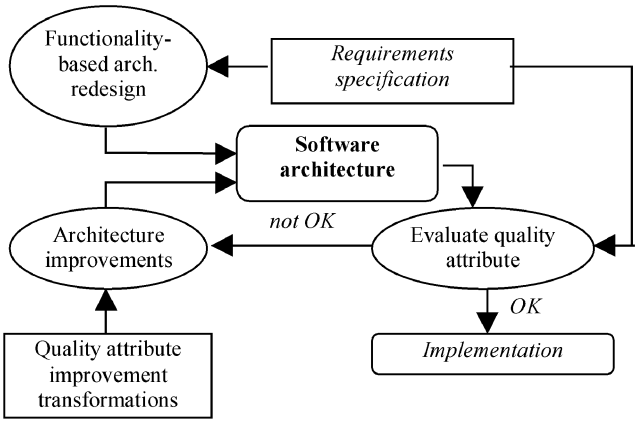

Fig. 7. SBAR activities. SA analysis and design [8].

Quality attributes. SBAR focuses on multiple software qualities. A number of quality attribute research communities have proposed their own methods for developing real-time [39], high performance [50], and reusable systems [24]. All these methods focus on a single quality attribute and treat all others as having secondary importance, if any at all. SBAR considers these approaches unsatisfactory because a balance of various quality attributes is needed in the design of any realistic system.

Stakeholders' involvement. SBAR does not require the involvement of many stakeholders. The evaluator is the designer of the SA.

SA description. A particularity of this method is that for assessing the architecture of the existing system, the system itself can be used. SBAR uses a detailed design of SA.

Method's activities. The assessment process consists of defining a set of scenarios for each software quality, manually executing the scenarios on the architecture and interpreting the results (Fig. 7). The method can be performed in a complete or statistical manner. In the first approach, a set of scenarios is defined and combined together, they cover the concrete instances of the software quality. If all scenarios are executed without problems, the quality attribute of the architecture is optimal. The second approach is to define a set of scenarios that makes a representative sample without covering all possible cases. The ratio between scenarios that the architecture can handle and scenarios not handled well by the architecture provides an indication of how well the architecture fulfils the software quality requirements. Both approaches obviously have disadvantages. A disadvantage of the first approach is that it is generally impossible to define a complete set of scenarios. The definition of a representative set of scenarios is the weak point in the second approach since it is unclear how one decides that a scenario set is representative. The results from each analysis of the architecture and scenario are summarized into overall results, e.g., the number of accepted scenarios versus the number not accepted. SBAR

TABLE 4
SBAR Method's Results Organization

Quality attribute	*Scenario*	*Iteration number*			
		0	*1*	...	*n*
QA1	A1	-			+
	A2	-			+
	…	-	-	-	+
QA2	B1	+	+	+	+
	B2	-	-	-	-
	…	-	-	+	+
…	….	…	…	…	…

provides guidelines for SA improvements. A structure similar to Table 4 is organized to express the results. Design and analysis combination is performed for a number of iterations until most of the scenarios per each quality attribute are satisfied (+).

Reusability of the existing knowledge base. SBAR does not consider this issue.

SBAR has been validated for a measurement software system.

3.7 Architecture Level Prediction of Software Maintenance (ALPSM)

Specific goal. ALPSM analyzes maintainability of a software system by looking at the impact of scenarios at the SA level [10]. Similar to the software maintenance community [11], it uses the size of changes as a predictor for the effort needed to adapt the system into a scenario.

The included evaluation technique. ALPSM defines a maintenance profile, like a set of change scenarios representing perfective and adaptive maintenance tasks. A scenario describes an action or a sequence of actions that might occur as related to the system. Hence, a change scenario describes a certain maintenance task.

Stakeholder's involvement. Only the designer is involved in the method activities.

SA description. ALPSM is applied to the final version of SA.

Method's activities. The method has a number of inputs: the requirements statement, the description of the architecture, expertise from software engineers and possibly historical maintenance data (Fig. 8). ALPSM consists of the following six steps:

1. the identification of categories of maintenance tasks,
2. synthesis scenarios,
3. assignation of a weight to each scenario,
4. estimation of the size of all elements,
5. scripting the scenarios, and
6. calculation of the predicted maintenance effort.

The first step formulates classes of expected changes based on the application or program description, then for each of the maintenance tasks, a representative set of scenarios is defined. The scenarios are assigned a weight based on their probability of occurring during a particular time interval. To be able to assess the size of changes, the size of all components of the system is determined. One of the three techniques can be used for estimating the size of components: using the estimation technique of choice, an

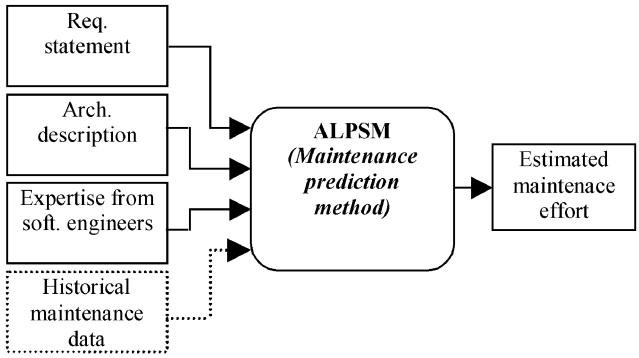

Fig. 8. ALPSM inputs and result.

adaptation of an object-oriented metric or, when historical data from similar applications or earlier releases is available, existing size data can be used and extrapolated to new components. The total maintenance effort is predicted by summing up the size of the impact of the scenarios multiplied by their probability. The size of the impact of each scenario realization is calculated by determining the components that are affected and to what extent will they be changed.

Reusability of a knowledge base is not considered, but historical data from similar applications or earlier releases are needed. Previous data are extrapolated to new components.

The method has been applied to a haemodialysis system.

3.8 A Software Architecture Evaluation Model (SAEM)

The evaluation process of the quality requirements of the SA is rigorously formalized, especially in relation to metrics in the model described in [18]. A quality model based on standard software quality assessment process [23] is chosen and a conceptual framework that relates quality requirements, metrics, and internal attributes of the SA and the final system is proposed. The elements required for quality evaluation of a software system, based on standard specification, are a quality model, a method for evaluation, metrics, and the supporting tools.

Specific goal of the method. SAEM establishes the basis for the SA quality evaluation and prediction of the final system quality.

The evaluation technique. SAEM tries to define metrics of quality based on the goal-question-metric (GQM) technique. The goal of metrics is to discover whether certain attributes meet the values specified in the quality specification for each software characteristic.

The quality attributes. The quality specification is divided into external and internal categories. The external quality expresses the user's view and the internal quality expresses the developer's view. The internal quality attributes are composed of *special elements* (such as functional elements or data elements) denoting quality characteristics and *intrinsic properties* resulting from the development process (such as size, modularity, complexity, coupling, and cohesion). It is necessary to establish a relative importance between internal attributes and their values; QFD [47] is recommended as a suitable technique for this purpose.

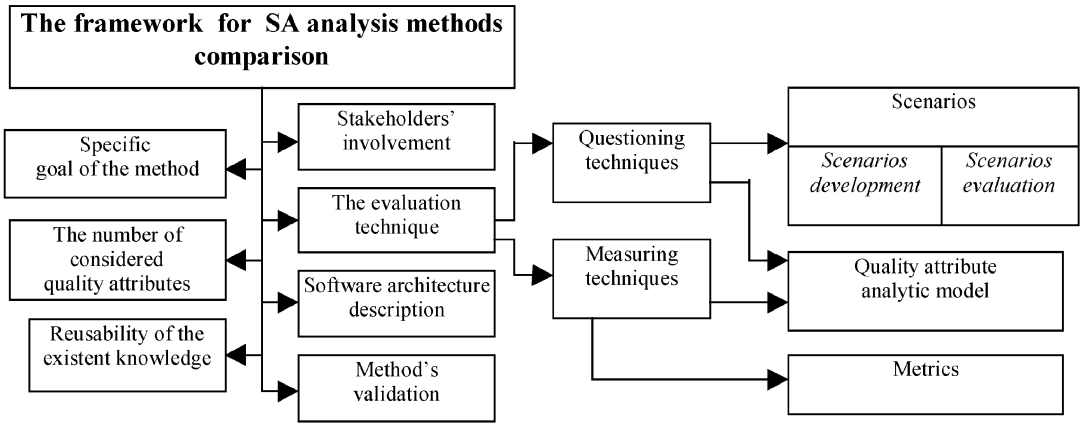

Fig. 9. Main elements of the comparison framework.

The stakeholders' involvement. Experts' knowledge and a company's accumulated data are used in the mapping of quality requirements to internal attributes.

SA description. SA is considered from two different viewpoints, that of the developer and that of the user. Therefore, the SA is either a final product or an intermediate in the software system development process. The architecture development process constrains the internal attributes, so the result of the measurement process can improve architecture as a form of feedback. The architecture description language (ADL) model should have attached questioning or inspection techniques (such as SA models walkthrough) to detect the presence or absence of special elements. The intrinsic properties can only be detected by measuring techniques applied to the SA representation formalized through an ADL.

Method's activities. SAEM gives a quality evaluation model based on data collection, measurement, and analysis of the results. The analysis process is divided into external and internal processes and is adapted to users' or developers' views. The specified quality requirements are mapped to internal attributes that will be present on the SA, based on the experts' knowledge and company accumulated data.

Reusability of the existing knowledge base is not considered. However, the evaluation model assumes the existence of a previous internal quality specification, which defines the expected internal attributes with their values and their evaluation procedure.

SAEM has not been validated on any software system.

4 Discussion

The purpose of this discussion is to offer guidelines related to the selection of the most suitable method for an architecture assessment process. Fig. 9 presents the main issues examined. The opening part focuses on a study that identifies the collective goal and how this goal is divided in each of the analysis methods. Then, several classifications of the methods are established. Included evaluation techniques, the number of quality attributes, the stakeholders' involvement, and SA description, or when the method is applied in the architecture-based development process are the main criteria of classification. To maintain a pertinent discussion in exemplification, we consider only the most representative methods. Common activities such as scenarios development and evaluation with their different approaches about when to stop generating scenarios and how the scenarios' impact on a considered architecture is evaluated are identified in the scenario-based analysis methods. The final part discusses the special case of the evolution of ATAM from SAAM and how the existing knowledge is reused by the analysis methods. Finally, we conclude with a summary of the considered SA analysis methods.

4.1 Appropriateness Study—Methods' Specific Goals

Objective views are considered a basis for establishing which analysis method is most suitable for an architecture assessment process. Although each method has its particularity in the definition of its objectives, we can identify in all of them a collective goal, which is the prediction of the quality of a system before it has been built. In each method, this goal is reflected under different angles and perspectives. The reflections are oriented to: guide the inspection of the SA, focusing on potential trouble spots (SAAM, ESAAMI); risk assessment (SAAMCS); evaluate the potential of the designed SA to reach the software quality requirements (SBAR, SAAMER); predict one quality attribute of a software system based on its architecture (ALPSM); establish the basis for SA evaluation and prediction of the final system quality (SAEM); locate and analyze trade-offs in an SA, for these are the areas of highest risk in an architecture (ATAM). The collective and particular characteristics of the goals lead to similarities and differences between all these presented methods.

4.2 Classifications of the Methods

Based on the evaluation techniques, we can establish a possible classification of the methods considering the techniques they use. From this point of view, some of the methods are: purely scenario-based, like SAAM; scenario-based and attribute model-based analysis technique, like ATAM; proposing various evaluation techniques depending on the attribute, like SBAR, and related to metrics, like SAEM. A quality model of attributes for quantitative evaluation is treated during the assessment process in two of the methods, but from this angle we identified different approaches. SAEM tries to define metrics based on the GQM technique, while ATAM considers that analysis

techniques indigenous to the various quality attribute communities can provide a foundation for performing SA evaluation. It is not necessary to invent attribute-specific techniques and metrics, but to integrate existing ones into systematic methods. ATAM provides flexibility in the integration of the best individual theoretical model of each considered attribute.

Based on the considered number of quality attributes, some methods are centered on the evaluation of a single quality attribute. However, for a better understanding of the strengths and weaknesses of a complex real system and its parts, the performance of a multiattribute analysis is required. An important feature revealed by studying the analysis methods is the number of quality attributes a method focuses on. We can distinguish multiple quality attributes (ATAM, SBAR). For example, ATAM considers the architectural elements where multiple attributes interact. There are also single quality attributes (SAAM) and a specific quality model (SAEM).

Based on stakeholders' involvement, although it is recognized that the involvement in the evaluation process of all the stakeholders facilitates communication between them, not all the methods consider their presence mandatory. ALPSM differs from SAAM in that it does not involve all stakeholders and, thus, requires less resources and time. Instead, it provides an instrument for the software architects that allows them to repeatedly evaluate the architecture during design. Due to the need of a stakeholder's commitment, this method could be used in combination with SAAM. In SAAM and ATAM, architecture is evaluated by the analysts in cooperation with stakeholders prior to the detailed design, while, in SBAR, architecture is evaluated on a detailed design for reengineering without the stakeholders' involvement, although at the same time posing typical quality questions.

When is the method applied? This question gets different responses when considering the architecture-based development process. A similar approach, which combines architecture analysis and design into an iterative improvement process, could be identified in ATAM and SBAR. But, while SBAR includes guidelines on how to transform the architecture in order to meet certain quality requirements, ATAM concentrates on identifying sensitivities and tradeoff points. However, ATAM could also be applied to the evaluation on the SA final version. SAAM, SAAMCS, ESAAMI, and SAAMER are also applied to the final version of the SA. ALPSM is applied during the design to predict adaptive and perfective software maintenance. SAEM is applied to the final version, but, here, it should be noted that the evaluation model considers SA from two different viewpoints, the developer's and the user's. Therefore, the SA is either a final product or an intermediate one in the system development process. The rigorous ambition of SAEM makes it hard to believe that it will be suitable for usage in an iterative SA design process.

4.3 Common Activities and Different Approaches in Scenario-Based Methods

The activities of the methods differ in *complexity* and *granularity/aggregation level*. Complexity represents the difficulty, measured in time or the number of other tools or documents needed to perform that activity. All the methods are performed manually and, for the moment, there is no requirement for any software tools. Documents are necessary in some of the methods and are contained in a reusable knowledge base. Granularity/aggregation level means that an activity may represent a group of subactivities or a phase that is divided into steps. For example, SAAMER defines a framework of four activities and one of them includes all the activities of SAAM. ATAM also consists of four phases, each one with multiple steps.

Scenario-based assessment is particularly appropriate for qualities related to software development. Software qualities such as maintainability, reusability, modifiability, adaptability, and portability can be expressed very naturally through change scenarios. As Poulin [46] concluded when considering reusability, no predominant approach for assessing this quality attribute exists. The use of scenarios for evaluating architectures is recommended as one of the best industrial practices [1]. To this end, we discuss, in the following, different proposals for scenarios development and scenario impact evaluation.

Scenarios development. A common activity of scenario-based methods is scenarios development. We identified different solutions that try to answer to the question, "when to stop generating scenarios?" during this activity. *SAAM* considers that the set of scenarios is complete when the addition of a new scenario no longer disturbs the design. Scenarios are also elicited considering all the stakeholders' opinions. In *SBAR*, two approaches are discussed. One is to define a complete set, which is generally impossible. The other is to define a representative set, which has the weak point of how to define which is the representative set. The last one is based only on the creativity and subjectivity of the software engineer. *SAAMCS* considers that the relevant scenarios are those that are possibly complex to realize. A two-dimensional framework diagram (five categories of complex scenarios, four sources of changes) that may help to discover complicated scenarios is defined. *SAAMER* defines a practical two-steps procedure. In the first step, a coverage guarantee is obtained. The scenarios are identified and clustered based on the objectives and domain experts' knowledge and the coverage is checked against the objectives of stakeholders, architecture, and quality. The second step validates the balance of scenarios with respect to the objective based on the QFD technique. The decision to develop more scenarios is made based on comparison against one of a calculated imbalance factor for each quality attribute. *ATAM* uses a set of *standard quality attribute-specific questions* to ensure proper coverage of an attribute by the scenarios. The boundary conditions should be covered. A standard set of quality-specific questions gives one the possibility to elicit the information needed to analyze that quality in a predictable, repeatable fashion. *ALPSM* defines a representative set of scenarios for each expected maintenance task.

Scenario Evaluation. There are variances in the evaluation of the scenarios' effects on the considered architecture. *SAAM* investigates which architectural elements are affected by each scenario. The cost of the modifications associated with each indirect scenario is estimated by relationships—listing the components and the connectors that are affected and then counting the number of changes. In *ALPSM*, the effort needed to implement the scenario is predicted by estimating the size of the components and the extent to which they are affected. This activity may need

historical maintenance data. *SAAMCS* defines and uses a measurement instrument to express the effect of scenarios. The instrument indicates the impact of a scenario, whether multiple owners are involved and whether it leads to version conflicts. In *SAAMER*, a classification and generalization of the architectural elements facilitates the estimation of the cost or effort required for changes to be made. The required changes, specified in scenarios and domain experts' experiences, suggest how each of the objectives or the risks for the systems evolution or reuse across applications could be supported. Finally, in *SBAR*, the evaluation can be performed in a complete or statistical manner. The optimality of a quality attribute could be obtained using the former approach and the fulfilment of a quality attribute using the latter one.

4.4 A Discussion of Methods' Evolution

A special case of qualitative and quantitative progress is observed at ATAM. Considering the uses of scenarios, ATAM is based on SAAM. Unlike SAAM, which focuses on the use of scenarios for architectural modifiability evaluation, ATAM focuses on finding trade-off points in the architecture from the perspective of quality requirements of the product. In addition, ATAM prescribes formal or informal analytic models for assessing the quality attributes of the system, but relies on the existence of such techniques for the quality attributes relevant to each case. In the case of modifiability analysis, ATAM builds an informal model like SAAM with inspection and review methods. Scenario interactions are interpreted as sensitivity points.

4.5 The Reusability of an Existing Knowledge Base

Similarities at a coarse-grain level could also be identified between ATAM and ESAAMI. Both methods are based on SAAM. Considering the reusability of the existing knowledge, ATAM uses ABASs and ESAMI proposes packages of analysis templates and reusable architectures. However, when we talk about systematization of information, there is no possible comparison. ESAAMI allows making domain-respective architecture-specific experience available in an intuitive form, while ATAM is anchored in a very well-structured knowledge base of quality attributes communities and architectural styles. ABASs provide a set of prepackages of analyses and questions including known solutions to commonly recurring problems and known difficulties in employing those solutions. ATAM is based on a set of materials that describe many of the evaluation artifacts, like ABASs, quality attribute-specific questions that aid the evaluator in probing an architecture and questions that aid the analyst in gathering the information needed to build an analytic model of the quality attribute.

4.6 About Choosing Analysis Methods in Practice

The selection of a suitable method depends on how well each comparison element fits into the problem context. It is not the purpose of this survey to suggest a ranking list of analysis methods to the practitioners, but to give an understanding of how the methods differ.

A summary of this discussion is depicted in Table 5 and 5a.

In practice, one of the purposes for using a software architecture analysis method is to decrease the costs caused from corrections and to increase quality of products. For that reason, a method 1) should be able to be used in an early or middle phase of the SA design process (smaller and cheaper errors), 2) should support all possible quality attributes or as many as possible (the scope of use), and 3) should be easy to apply and integrate to the design process (it takes less time to apply, by a designer). If all these criteria are supported by a method, it can be selected and applied in practice. From Table 5, we can see that, except SBAR (applied on a reengineered SA design) and ESAAMI (applied on a selected architecture design from a domain database), all the others may fulfill 1). Point 2 is supported by ATAM and SBAR and point 3 by SBAR, ALPSM, and ATAM (when analysis is performed by a designer). The result is that ATAM satisfies as many as possible of the proposed criteria.

Only one selection criterion could be insufficient to indicate the most suitable method for a defined purpose. The included evaluation techniques, the ease with which the method's activities are performed, and the existence of a knowledge base may represent other criteria that have to be considered in the selection process. An important element to think about is how well and in what software domain a method has been validated in practice.

5 Conclusions and Future Work

This survey has shown the real level of the research at this moment, in this domain, by presenting and discussing eight of the most representative architecture analysis methods. This section is organized to reveal the general progress, existing problems, and future work for improvement and refinement.

5.1 Progress Identification and Methods Improvement Techniques

Progress in risk assessment. The purpose of the evaluation is to analyze the architecture to identify potential risks, by predicting the quality of the system before it has been built. Regarding potential risks identification, the reflections of this general goal have been distinguished in all the studied methods. In this sense, the uses of change scenarios and scenario interaction reveal potential problem areas in the architecture. The degree of modification captured when evaluating a system's response to a scenario represents a measured risk. The complexity of scenarios is also an important factor for risk assessment. The required changes and domain experts' experiences represent another modality of suggestion of how the system could support the risk levels for evolution or reuse. The chances of surfacing decisions at risk are optimized by using exploratory scenarios. The potential risk is also minimized by analyzing attribute interactions. Iterative methods promote analysis at multiple resolutions as a means of minimizing risk to an acceptable level of time and effort. Areas of high risk are analyzed more profoundly (simulated, modeled, or prototyped) than the rest of the architecture. Each level of analysis helps to determine where to analyze more deeply in the next iteration.

A possible combination of methods. Looking at the existing analysis techniques, the possibility of combining a coarse-grain and broad technique with a fine-level one would provide an improved result, but the costs of time and effort would also be increased. Scenario-based analysis

TABLE 5
Software Architecture Analysis Methods

Method / *Comparison element*	SAAM	SAAMCS	ESAAMI	SAAMER
The evaluation technique	Scenarios.	Scenarios.	Scenarios.	Scenarios.
Quality attributes	Modifiability	Flexibility	Similar to SAAM.	Evolution and reusability.
Stakeholders' involvement	All	All	All	All
The phase of SA design	In the final version of the SA.	In the final version of the SA.	In the final version of the SA.	In the final version of the SA.
When to stop generating scenarios?	If the addition of a new scenario no longer perturbs the design.	Defines a framework to discover all the complicated scenarios.	Similar to SAAM, but it considers protoscenarios, too.	Uses a practical two-step procedure.
Scenarios impact evaluation	Relationships.	Relationships, owners, versions.	Similar to SAAM.	Estimates the cost required for a change to be made.
Reusability of the existing knowledge	Not considered.	Not considered.	Analysis templates and reusable SAs in the domain.	Not considered.

techniques can be combined with a specific analysis technique for quality attributes. For example, a scenario may identify a critical path of execution, which can then be examined in detail using a real-time analysis method like rate monotonic analysis (RMA) [2], [31], or other analysis techniques for scrutinizing the dynamic properties of an application.

Metrics—more precise techniques in evaluating attributes in terms of architecture. Most of the researchers in the domain consider metrics to be a more precise technique in evaluating attributes in terms of architecture [28], [41]. Metrics specification must contain the selected measure for a quality attribute, a measurement scale, and a set of methods for measurement. Two approaches could be identified: to adapt existing metrics [9] or to define new ones [18]. The adaptation of object-oriented metrics which were validated as good predictors of software maintenance [38] is required because the metrics suite uses data that can only be collected from the source code and, at the

TABLE 5a
TABLE 5 (cont.)

Method / *Comparison element*	ATAM	SBAR	ALPSM	SAEM
The included evaluation technique	Integrates existent questioning and measuring techniques	Depends on the attribute: scenarios, mathematical modeling, simulators, objective reasoning	Scenarios.	Metrics. Different metrics based on GQM technique.
Quality attributes	Multiple quality attributes	Multiple quality attributes	Maintainability	A quality model.
Stakeholders' involvement	All or Designer only	Designer	Designer	Not applied
The phase of SA design	In the final version or combined with the SA design into an iterative improvement process.	Combined with the SA design into an iterative improvement process and re-engineering.	During design to predict adaptive and perfective software maintenance.	In the final version of the SA.
When to stop generating scenarios?	Uses a standard set of quality attribute-specific questions.	Defines a complete set or a representative set of scenarios.	Defines a set of scenarios for each expected maintenance task.	Not applied
Scenarios impact evaluation	Similar to SAAM when applied.	Optimized or fulfilled.	Estimates the size of the components and the extent to which they are affected.	Not applied.
Reusability of the existing knowledge base	A set of pre-packages analyses and questions with known solutions.	Not considered.	Not considered.	Not applied.

architecture level, no prototype or source exists. Considering the other approach, GQM [6] is a good technique to define new metrics following a certain reasoning process. The main activities of GQM are: to define a goal in terms of purpose, perspective, and environment; to establish the questions that indicate the attributes related to the goal; and to answer to each question. The purpose is related to the SA evaluation, indication, and comparison, and the end product quality prediction. The perspective depends on the aims of the assessment and it is closely related to the role of evaluation staff: a developer, user, management, and maintainer. There are two suitable environments: the SA representation considered as an intermediate design product or as an end product in itself.

QFD—a technique to be considered. QFD is a technique that should be considered as a future research topic. It has been used in SAAMER and it is recommended by SAEM to developers, in order to establish the relative importance between attributes and their values. Equally, this technique is seen as useful in the process of formalization of the relationship between internal quality attributes and the quality characteristics/subcharacteristics which must be studied for specific application domains, development processes, and ADL.

5.2 Open Problems and Future Work

Scenarios and quality attributes naming problems. One problem with scenario-based analysis is that the result and expressiveness of the analysis are dependent on the selection of the scenarios and their relevance for identifying critical assumptions and weaknesses within the architecture. There is no fixed minimum number of scenarios, the evaluation of which guarantees that the analysis is meaningful. According to this, the definition of a set of complex scenarios and a two-dimensional framework is a solution, but a future study of this set completeness and on the relative importance of each of the framework cells is needed. The idea of using an instrument which should include all aspects relevant to the complexity of changes is original and useful, but the measures must be comparable to allow results' interpretation.

Future studies are needed in order to investigate how domain knowledge and the degree of expertise affect the coverage of the selected scenarios. By the same token, quality attributes prediction methods could be improved by studying their sensitivities for different variations of the inputs and how significant the used assumptions parameters are for the results. For instance, how sensitive ALPSM is to the representative sample of the maintenance scenario profile, or how critical the size estimation for the results is.

An examination of the existent methods reveals a lack of understanding of quality attributes in the software engineering community at the moment. The same interpretation but with different attribute names could be identified for flexibility [37], which has the same meaning as modifiability in [26] or maintainability in [10].

Future work for methods improvement and refining. Until now, SAAM has been the only method that has appeared in a book [7]. This is a confirmation of its maturity. SAAM has been used for different quality attributes like modifiability, performance, availability, and security. It has also been applied in several domains. This is another validation of its completeness. The other methods are still young and are undergoing refinement and improvement. Future work is needed to evaluate the effects of their various usages and to create a repeatable method based on repositories of scenarios, screening, and elicitation questions (ATAM). In this respect, ABASs and qualitative analysis heuristics are developing. Building a handbook of ABASs requires collection, documentation, and testing of many examples of problems, quality attributes measures, stimuli, and parameters.

The extension of the reengineering method for more nonfunctional requirements and the application of the method in more industrial case studies are the main future objectives of SBAR. The authors of this method consider it important to obtain a reasonable balance between the different quality requirements in the top-level architectural design. A small taxonomy is defined for performance and modifiability, and eight design guidelines are formulated. Each guideline is associated with a quality requirement in the taxonomy. Future work is desirable, in order to extend these guidelines to other quality requirements.

A stronger methodical integration in the development process is also required. ESAAMI needs to provide complete support for the reuse-based and architecture-driven development approaches. Integrating the technique into reuse-based and architecture-centric development processes should provide a refinement of the method.

REFERENCES

[1] G. Abowd, L. Bass, P. Clements, R. Kazman, L. Northop, and A. Zaremski, "Recommended Best Industrial Practice for Software Architecture Evaluation," Technical Report, CMU/SEI-96-TR-025, 1997.

[2] A. Alonso, M. Garcia-Valls, and J.A. de la Puente, "Assessment of Timing Properties of Family Products," *Proc. Second Int'l ESPRIT ARES Workshop,* pp. 161-169, Feb. 1998.

[3] M. Barbacci, M. Klein, and C. Weinstock, "Principles for Evaluating the Quality Attributes of a Software Architecture," Technical Report, CMU/SEI-96-TR-036, ESC-TR-96-136, 1997.

[4] M. Barbacci, M. Klein, T. Longstaff, and C. Weinstock, "Quality Attributes," Technical Report CMU/SEI-95-TR-021, ESC-TR-95-021, 1995.

[5] M. Barbacci, S. Carriere, P. Feiler, R. Kazman, M. Klein, H. Lipson, T. Longstaff, and C. Weinstock, "Steps in an Architecture Tradeoff Analysis Method: Quality Attribute Models and Analysis," Technical Report, CMU/SEI-97-TR-029 ESC-TR-97-029, 1998.

[6] V.R. Basili and H.D. Rombach, "Goal/Question/Metric Paradigm: The TAME Project: Towards Improvement-Oriented Software Environments," *IEEE Trans. Software Eng.,* vol. 14, no. 6, 1988.

[7] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice.* Reading, Mass.: Addison-Wesley, 1998.

[8] P.O. Bengtsson and J. Bosch, "Scenario-Based Architecture Reengineering," *Proc. Fifth Int'l Conf. Software Reuse (ICSR 5),* 1998.

[9] P.O. Bengtsson, "Towards Maintainability Metrics on Software Architecture: An Adaptation of Object Oriented Metrics," *Proc. First Nordic Workshop Software Architecture (NOSA '98),* Aug. 1998.

[10] P.O. Bengtsson and J. Bosch, "Architecture Level Prediction of Software Maintenance," *Proc. Third European Conf. Software Maintenance and Reeng.,* pp. 139-147, Mar. 1999.

[11] R.S. Arnold and S.A. Bohner, *Software Change Impact Analysis.* Los Alamitos, Calif.: IEEE Computer Society, 1996.

[12] J. Bosch and P. Molin, "Software Architecture Design: Evaluation and Transformation," *Proc. IEEE Eng. of Computer Based Systems Symp. (ECBS '99),* Dec. 1999.

[13] S. Bot, C.-H. Lung, and M. Farrell, "A Stakeholder–Centric Software Architecture Analysis Approach," *Proc. Int'l Software Architecture Workshop (ISAW 2),* 1996.

[14] L. Bratthall and P. Runeson, "A Taxonomy of Orthogonal Properties of Software Architecture," *Proc. Second Nordic Software Architecture Workshop (NOSA '99),* 1999.

[15] L.C. Briand, S. Morasca, and V.R. Basili, "Measuring and Assessing Maintainability at the End of High Level Design," *Proc. IEEE Conf. Software Maintenance,* 1993.

[16] F. Buschmann, R. Meunier, P. Sommerland, and M. Stal, *Pattern-oriented Software Architectures, a System of Patterns.* Chichester: Wiley & Sons, 1996.

[17] R. Day, *Quality Function Deployment. Linking a Company with Its Customers.* Milwaukee, Wisc.: ASQC Quality Press, 1993.

[18] J.C. Duenas, W.L. de Oliveira, and J.A. de la Puente, "A Software Architecture Evaluation Model," *Proc. Second Int'l ESPRIT ARES Workshop,* pp. 148-157, Feb. 1998.

[19] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns—Elements of Reusable Object-Oriented Software.* Reading, Mass.: Addison-Wesley, 1995.

[20] A. Iannino, "Software Reliability Theory, *Encyclopedia of Software Eng.,* J.J. Marciniak, ed., vol. 2, pp. 1237-1253, 1994.

[21] *IEEE Standard Glossary of Software Engineering Terminology.* IEEE Std. 610.12-1990, 1990.

[22] IEEE Standard 1061-1992, *Standard for Software Quality Metrics Methodology,* New York: Institute of Electrical and Electronics Engineers, 1992.

[23] ISO/IEC91—Int'l Organization of Standardisation and Int'l Electrotechnical Commission, *Information Technology—Software Product Evaluation—Quality Characteristics and Guidelines for Their Use,* ISO/IEC 9216: 1991(E), 1991.

[24] *Software Reuse, a Holistic Approach.* E. Karlsson ed., Chichester: Wiley & Sons, 1995.

[25] R. Kazman, M. Barbacci, M. Klein, S.J. Carriere, and S.G. Woods, "Experience with Performing Architecture Tradeoff Analysis," *Proc. Int'l Conf. Software Eng. (ICSE '99),* pp. 54-63, May 1999.

[26] R. Kazman, G. Abowd, L. Bass, and P. Clements, "Scenario-Based Analysis of Software Architecture," *IEEE Software,* pp. 47-55, Nov. 1996.

[27] R. Kazman, G. Abowd, L. Bass, and M. Webb, "Analyzing the Properties of User Interface Software Architectures," Technical Report, CMU-CS-93-201, Carnegie Mellon Univ., School of Computer Science, 1993.

[28] R. Kazman, L. Bass, G. Abowd, and M. Webb, "SAAM: A Method for Analyzing the Properties of Software Architectures," *Proc. 16th Int'l Conf. Software Eng.,* pp. 81-90, 1994.

[29] R. Kazman, M. Klein, M. Barbacci, H. Lipson, T. Longstaff, and S.J. Carrière, "The Architecture Tradeoff Analysis Method," *Proc. Fourth Int'l Conf. Eng. of Complex Computer Systems (ICECCS '98),* Aug. 1998.

[30] R. Kazman, S.J. Carriere, and S.G. Woods, "Toward a Discipline of Scenario-Based Architectural Engineering," *Annals of Software Eng.,* vol. 9, 2000, http://www.cgl.uwaterloo.ca/~rnkazman/SE-papers.html.

[31] M. Klein, T. Ralya, B. Pollak, R. Obenza, and M. Gonzales Harbour, *A Practitioner's Handbook for Real-Time Analysis.* Boston: Kluwer Academic, 1993.

[32] M. Klein, R. Kazman, L. Bass, S.J. Carriere, M. Barbacci, and H. Lipson, "Attribute-Based Architectural Styles," *Proc. First Working IFIP Conf. Software Architecture (WICSA 1),* pp. 225-243, Feb. 1999.

[33] P.B. Krutchen, "The 4+1 View Model of Architecture," *IEEE Software,* pp. 42-50, Nov. 1995.

[34] N.H. Lassing, D.B.B. Rijsenbrij, and J.C. van Vliet, "Flexibility in ComBAD Architecture," *Proc. First Working IFIP Conf. Software Architecture (WICSA 1),* Feb. 1999.

[35] N. Lassing, D. Rijsenbrij, and H. van Vliet, "On Software Architecture Analysis of Flexibility, Complexity of Changes: Size Isn't Everything," *Proc. Second Nordic Software Architecture Workshop (NOSA '99),* pp. 1103-1581, 1999.

[36] N. Lassing, D. Rijsenbrij, and H. van Vliet, "The Goal of Software Architecture Analysis: Confidence Building or Risk Assessment," *Proc. First Benelux Conf. State-of-the-art of ICT Architecture,* 1999.

[37] N. Lassing, D. Rijsenbrij, and H. van Vliet, "Towards a Broader View on Software Architecture Analysis of Flexibility," *Proc. Asian-Pacific Software Eng. Conf. (APSEC '99),* 1999.

[38] W. Li and S. Henry, "Object-Oriented Metrics that Predict Maintainability," *J. Systems and Software,* vol. 23, no. 2, pp. 111-122, 1993.

[39] J.W.S. Liu and R. Ha, "Efficient Methods of Validating Timing Constraints," *Advances in Real-Time Systems,* S.H. Son ed., pp. 199-223, 1995.

[40] L. Lundberg, J. Bosch, D. Häggander, and P.O. Bengtsson, "Quality Attributes In Software Architecture Design," *Proc. IASTED Third Int'l Conf. Software Eng. and Applications,* pp. 353-362, Oct. 1999.

[41] C. Lung, S. Bot, K. Kalaichelvan, and R. Kazman, "An Approach to Software Architecture Analysis for Evolution and Reusability," *Proc. CASCON '97,* Nov. 1997.

[42] J.A. McCall, "Quality Factors," *Encyclopedia of Software Eng.,* J.J. Marciniak ed., vol. 2, pp. 958-971, 1994.

[43] G. Molter, "Integrating SAAM in Domain-Centric and Reuse-Based Development Processes," *Proc. Second Nordic Workshop Software Architecture (NOSA '99),* pp. 1103-1581, 1999.

[44] D. Parnas, "On the Criteria to be Used in Decomposing Systems into Modules," *Comm. ACM,* vol. 15, no. 12, pp. 1053-1058, 1972.

[45] D. Perry and A. Wolf, "Foundation for the Study of Software Architecture," *SIGSOFT Software Eng. Notes,* vol. 17, no. 4, pp. 40-52, 1992.

[46] J.S. Poulin, "Measuring Software Reusability," *Proc. Third Conf Software Reuse,* Nov. 1994.

[47] B.M. Reed and D.A. Jacobs, *Quality Function Deployment for Large Space Systems.* Nat'l Aeronautics and Space Administration, 1993.

[48] P. Runeson and C. Wohlin, "Statistical Usage Testing for Software Reliability Control," *Informatica,* vol. 19, no. 2, pp. 195-207, 1995.

[49] M. Shaw and D. Garlan, *Software architecture. Perspectives on an Emerging Discipline.* Upper Saddle River, N.J.: Prentice Hall, 1996.

[50] C. Smith, *Performance Engineering of Software Systems.* Reading, Mass.: Addison-Wesley, 1990.

[51] C. Smith and L. Williams, "Software Performance Engineering: A Case Study Including Performance Comparison with Design Alternatives,," *IEEE Trans. Software Eng.,* vol. 19, no. 7, pp. 720-741, July 1993.

[52] W.P. Stevens, G.J. Myers, and L.L. Constantine, "Structured Design," *IBM Systems J.,* vol. 13, no. 2, pp. 115-139, 1974.

Liliana Dobrica received the MSc degree in 1991 in process control software engineering and PhD degree in 1998 in control systems from the Politehnica University of Bucharest, Romania. She is an associate professor in the Department of Control and Industrial Informatics of Faculty of Automation and Computers Science at the Politehnica University of Bucharest, Romania. In 2000, she joined Software Architectures Research Group, VTT Electronics, Oulu, Finland, where she was a postdoctoral research associate for nine months. Her research interests include software design and analysis for embedded, real-time, and distributed systems, software architecture, and product-line architecture, quality attributes analysis techniques, with emphasis on integrating quality attribute analysis techiques into the software development process. Her current research projects include modeling and analysis of software product-line architecture for middleware services domain. She has published several journal and conference technical papers.

Eila Niemelä received the MSc degree in information processing science from the University of Oulu, Finland, in 1995. Between 1995 and 1998, she worked as a researcher in the Software Architectures Group at VTT Electronics. She was a visiting researcher at Napier University, Edinburg, UK in 1998-99. Since October 1999, she has worked as a group manager in the Software Architectures Group of the Embedded Software research area. In 2000, she obtained the PhD degree in information processing science from the University of Oulu, a component framework of a distributed control systems family as a topic. Since 2001, she has worked as a research professor at VTT Electronics. She has published several conference papers about software architectures and components, as well as embedded middleware services. She is a member of the IEEE Computer Society.