Volume 1, No. 4, November 2010
Journal of Global Research in Computer Science
ISSN-2229-371X
REVIEW ARTICLE
Available Online at www.jgrcs.info

COMPARISON OF SOFTWARE ARCHITECTURE EVALUATION METHODS FOR SOFTWARE QUALITY ATTRIBUTES

L. S. Maurya[*1] and Himanshu Hora[2]

[*1]Associate Professor, [2]Assistant Professor
Shri Ram Murti Smarak College of Engineering & Technology, Bareilly (U.P.) India
lsmaurya@yahoo.com[1]
himanshuhora@gmail.com[2]

Abstract: Since the architecture of a software system constrains the quality attributes, the decisions taken during architectural design have a large impact on the resulting system. An architectural design method is presented that applies iterative evaluation of the software architecture in order to the quality requirements. Architecture evaluation is performed by using scenarios, simulation, mathematical modeling and experience-based reasoning. The software architecture has been keyed as an important part of a software system. Further, the software architecture impacts the quality attributes of a system, e.g., performance and maintainability. Therefore, methods for evaluating the quality attributes of software architectures are important. In this paper, we present a survey of software architecture evaluation methods. We concentrate on methods for evaluating one or several of the quality attributes performance, maintainability, testability, and portability. Based on a literature search and review of 76 articles, we present and compare ten evaluation methods. We have found that most evaluation methods only address one quality attribute, and very few can evaluate several quality attributes simultaneously in the same framework or method. Further, only one of the methods includes trade-off analysis. Therefore, our results suggest an altered research focus on software architecture evaluation methods than can direct several quality attributes and the possible trade-offs between different quality attributes.

Keywords: Software architecture, quality attributes, software system

INTRODUCTION

The software engineering discipline is becoming more widespread in industry and organizations due to the increased presence of software and software-related products and services in all areas. Simultaneously, this demands for new concepts and innovations in the development of the software. During the last decades, the notion of software architecture has evolved and today, software architecture is a key asset for any organization that builds complex software- intensive systems [5, 8, 34]. A software architecture is created early in the development and gives the developers a means to create a high level design for the system, making sure that all requirements that has to be fulfilled will be possible to implement in the system. There exist a number of definitions of software architecture with minor differences depending on domain and people's experience. However, most definitions share common characteristics that can be exemplified by looking at the definition by Bass et al. [5]:

"The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them." [5]

This means that the architecture describes which high level components a software system consists of as well as which responsibilities that these components have towards other components in the system. It also describes how these components are organized, both on a conceptual level as well as a decomposed detailed level since there can be an architectural structure inside components as well. Finally the architecture defines which interfaces the components present to other components and which interfaces and components that they use.

The architecture is created based on a set of requirements that it has to fulfill. These requirements are collected from the stakeholders of the system, e.g., users and developers. The functional requirements describe what the system should do, e.g., the functions that the system should provide to the users. Quality requirements describe a set of qualities that the stakeholders want the systems to have, e.g., how long time it may take to complete a certain operation, how easy it is to maintain the system. Other examples of quality attributes are availability, testability, and flexibility. In order to help software developers make sure that software architecture will be able to fulfill the quality requirements; several methods for evaluating software architectures have been proposed. In this paper we present a survey of software architecture evaluation methods. We focus our survey on methods that address one or more of the quality attributes performance, maintainability, testability, and portability. We think that this selection of quality attributes is relevant for development of software systems that will be used and maintained over a long period of time.

The methods are described and compared based on a set of criteria. There are related evaluations methods that we have chosen to exclude from our survey. One class of related evaluation methods are targeted for components and middleware, e.g., i-Mate [27]. These methods are excluded since they do not evaluate the whole architecture of a system. Further, we have exclude many formal methods, e.g., Promela/SPIN [16, 27], which are more targeted for evaluating correctness and consistency of an architecture but not those quality attributes that we are interested in. In addition, there are other factors then quality requirements that influence the architecture such as organizational, technical and product factors as well as risk management and project management issues. These factors and issues are not addressed since the majority of the found articles do not address these issues.

SOFTWARE ARCHITECTURE EVALUATION

Architecture evaluations can be performed in one or more stages of the software development process. They can be used to compare and identify strengths and weaknesses in different architecture alternatives during the early design stages. They can also be used for evaluation of existing systems before future maintenance or enhancement of the system as well as for identifying architectural drift and erosion.
Software architecture evaluation methods can be divided into four main categories, i.e., experience-based, simulation-based, mathematical modeling based. Methods in the categories can be used independently but also be combined to evaluate different aspects of software architecture, if needed [8].
Experience-based evaluations are based on the previous experience and domain knowledge of developers or consultants [2]. People who have encountered the requirements and domain of the software system before can based on the previous experience say if a software architecture will be good enough [8].
Simulation-based evaluations rely on a high level implementation of some or all of the components in the software architecture. The simulation can then be used to evaluate quality requirements such as performance and correctness of the architecture. Simulation can also be combined with prototyping, thus prototypes of architecture can be executed in the intended context of the completed system. Examples of methods in this group are Layered Queuing Network (LQN) [1] approaches and event-based methods such as RAPIDE [28, 29].
Mathematical modeling uses mathematical proofs and methods for evaluating mainly operational quality requirements such as performance and reliability [34] of the components in the architecture. Mathematical modeling can be combined with simulation to more accurately estimate performance of components in a system.
Scenario-based architecture evaluation tries to evaluate a particular quality attribute by creating a scenario profile that forces a very concrete description of the quality requirement. The scenarios from the profile are then used to step through the software architecture and the consequences of the scenario are documented. Several scenario based evaluation methods have been developed, e.g., Software Architecture Analysis Method (SAAM) [19], Architecture Trade-off Analysis Method (ATAM) [21], and Architecture Level Modifiability Analysis (ALMA) [6, 7].

QUALITY ATTRIBUTES

Software quality is defined as the degree to which software possesses a desired combination of attributes [17]. According to [8] the quality requirements that software architecture has to fulfill are commonly divided in two main groups based on the quality they are requesting, i.e., development and operational qualities. A development quality requirement is a requirement that is of importance for the developers work, e.g., maintainability, understandability, and flexibility. Operational quality requirements are requirements that make the system better from the users point of view, e.g. performance and usability. Depending on the domain and priorities of the users and developers, quality requirements can become both development and operational, such as performance in a real-time system.
A quality attribute can be defined as a property of a software system [5]. A quality requirement is a requirement that is placed on a software system by a stakeholder; a quality attribute is what the system actually presents once it has been implemented. During the development of the architecture it is therefore important to validate that the architecture has the required quality attributes, this is usually done using one or more architecture evaluations.

QUALITY ATTRIBUTES IN FOCUS

This survey focuses on software architecture evaluation methods that address one or more of the following quality attributes: performance, maintainability, testability, and portability. The IEEE standard 610.12-1990 [17] defines the four quality attributes as:

Maintainability: This is defined as:

"The ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment."

Maintainability is a multifaceted quality requirement. It incorporates aspects such as readability and understandability of the source code. Maintainability is also concerned with testability to some extent, as the system has to be re-validated during the maintenance.

Performance: Performance is defined as:

"The degree to which a system or component accomplishes its designated functions within given constraints, such as speed, accuracy, or memory usage."

There are many aspects of performance, e.g., latency, throughput, and capacity.

Testability: Testability is defined as:

"The degree to which a system or component facilitates the establishment of test criteria and the performance of tests to determine whether those criteria have been met."

We interpret this as the effort needed to validate the system against the requirements. A system with high testability can be validated quickly.

Portability: Portability is defined as:

"The ease with which a system or component can be transferred from one hardware or software environment to another."

We interpret this as portability not only between different hardware platforms and operating systems, but also between different virtual machines and versions of frameworks.

These four quality attributes are selected, not only for their importance for software developing organizations in general, but also for their relevance for organizations developing software in the real-time system domain in a cost effective way, e.g., by using a product-line approach. Performance is important since a system must fulfill the performance requirements, if not, the system will be of limited use, or not used. The long-term focus forces the system to be maintainable and testable, it also makes portability important since the technical development on computer hardware technology moves quickly and it is not always the case that the initial hardware is available after a number of years.

RELATED WORK

Surveying software architecture evaluation methods has, as far as we know, been done in four previous studies. In two of the cases, Dobrica and Niemelä [11] and Babar et al. [3], the software architecture evaluation methods are compared with each other in a comparison framework, specific for each study. The survey by Etxeberria and Sagardui [13] compares architecture evaluation methods with respect to the context of architectures in software product lines. The last survey, by Kazman et al. [20], does not address a large number of architecture evaluation methods but uses two evaluation methods as examples for illustrating how the methods fulfill a number of criteria the authors argue are highly needed for an architecture evaluation method to be usable.

The Dobrica and Niemelä survey [11], the earliest one, presents and compares eight of the "most representative", according to themselves, architecture evaluation methods. The discussion of the evaluation methods focus on 1) discovering differences and similarities and 2) making classifications, comparisons and appropriateness studies. The comparison and characterization framework in the survey comprises the following elements; the methods goal, which evaluation techniques are included in the method, quality attributes (what quality attributes and what number of quality attributes is considered), the software architecture description (what views are the foci and in which development phase), stakeholders' involvement, the activities of the method, support for a reusable knowledge base and the validation aspect of the evaluation method.

The objective of the Babar et al. survey [3] is to provide a classification and comparison framework by discovering commonalities and differences among eight existing scenario-based architecture evaluation methods. To a large extent, the framework comprises features that are either supported by most of the existing methods or reported as desirable by software architecture researchers and practitioners. The framework comprises the following elements; the method's Maturity stage, what definition of software architecture is required, process support, the method's activities, goals of the method, quality attributes, applicable project stage, architectural description, evaluation approaches (what types of evaluation approaches are included in the method?), stakeholders involvement, support for non-technical issue, the method's validation, tool support, experience repository, and resources required. The survey by Etxeberria and Sagarduia [13] addresses an evaluation framework for software architecture evaluation methods addressing software product-line architectures. Since the life span of a product-line architecture is longer than for ordinary software architectures evolution is one prioritized quality attribute that deserves extra attention in an evaluation. There exist other quality attributes as well, e.g. variability. The context of software product lines imposes new requirements on architecture evaluation methods and this is discussed by Etxeberria and Sagarduia and reflects their classification framework. The framework comprises the following elements; The goal of the method, attribute types (what domain engineering and application engineering quality attributes are addressed), evaluation phase (in the product line context the evaluation can take place on different phases in application engineering and domain engineering, respectively, as well as in a synchronization phase between the two), evaluation techniques, process description, the method's validation and relation to other evaluation methods.

The purpose of the last survey, by Kazman et al. [20], is primary to provide criteria that are important for an evaluation method to address, and not to compare existing evaluation methods. The authors argue for criteria addressing what it means to be an effective method, one that produces results of real benefit to the stakeholders in a predictable repeatable way, and a usable method one that can be understood and executed by its participants, learned reasonably quickly, and performed cost effectively. Thus, the survey ends up with the following four criteria: 1) Context and goal identification, 2) Focus and properties under examination, 3) Analysis Support, and 4) Determining analysis outcomes. The survey by Dobrica and Niemelä [11] provides an early, initial overview of the software architecture evaluation methods. This was followed up by the survey by Babar et al. [3] that presents a more detailed break-down (including requirements on detailed method activities etc.) and a more holistic perspective, e.g., process support, tool support. The survey by Kazman et al. [20] presents additional requirements on what a software architecture method should support. The software product-line context survey by Etxeberria and Sagarduia [13] addresses evaluation methods from a prescribed way of developing software. This perspective opened up some additional phases where an evaluation can take place and put product-line important quality attributes more in focus, e.g., variability and maintainability.

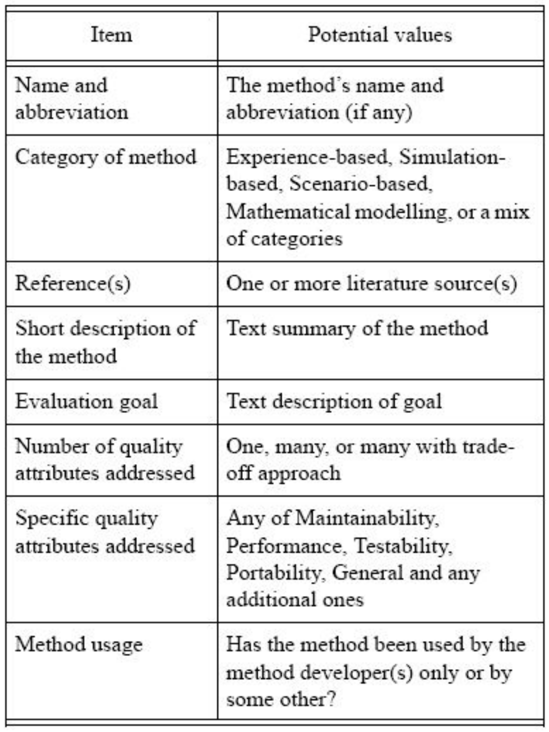
Our survey takes the perspective from a set of quality attributes that are of general importance for software developing organizations. This means that we are taking a more solution-oriented approach, i.e., we are focusing on finding knowledge about what existing evaluation methods can provide with respect to the identified quality attributes. We are not aiming at obtaining knowledge about general software architecture evaluation methods or pose additional requirements on the methods due to some completeness criteria or specific way of developing the software, as in the four performed surveys. We may add additional requirements on the evaluation method, but if that is the case, the requirements will have its origin from the four quality attributes addressed, performance, testability, maintainability and portability.

ARCHITECTURE EVALUATION METHODS

In this survey each of the software architecture evaluation methods will be described according to a pre-defined template. The template structures the description of the architecture according to the following elements: Name and abbreviation (if any), Category of method, Reference(s) where the method are described in detail, Short description of the method, Evaluation goal of the method, How many quality attributes the method addresses, (one, many, or many where trade-off approaches exist), What specific quality attributes the method address (or if

it is a more general evaluation method) and finally, the usage of the method. Table 1 summarizes the template with indication of potential values for each element. The initial selection of research papers was made by searching through Compendex, Inspec, and IEEE Xplore. The search Compendex and Inspec resulted in 194 papers, and the search in IEEE Xplore produced an additional 46 papers. The query used for the searched used the following keywords, "software architecture" and "any of evaluation, assessment or analysis" and "at least one of performance, maintainability, testability, or portability". The keywords where truncated and stemmed when possible. In total, we had 76 papers found from the database searches. We then eliminated duplicate papers and papers that did not fulfill our criteria of addressing one or more of the quality attributes performance, maintainability, testability, or portability.
After the screening we had about 25 papers that contained architecture evaluation methods and experience reports from their use. From these papers we have identified 10 methods and approaches that can be applied for architecture-level evaluation of performance, maintainability, testability, or portability.

Table-1: Method Description Template



Item	Potential values
Name and abbreviation	The method's name and abbreviation (if any)
Category of method	Experience-based, Simulation-based, Scenario-based, Mathematical modelling, or a mix of categories
Reference(s)	One or more literature source(s)
Short description of the method	Text summary of the method
Evaluation goal	Text description of goal
Number of quality attributes addressed	One, many, or many with trade-off approach
Specific quality attributes addressed	Any of Maintainability, Performance, Testability, Portability, General and any additional ones
Method usage	Has the method been used by the method developer(s) only or by some other?

SAAM — SOFTWARE ARCHITECTURE ANALYSIS METHOD

Software Architecture Analysis Method (SAAM) [19] is a scenario-based software architecture evaluation method, targeted for evaluating a single architecture or making several architectures comparable using metrics such as coupling between architecture components. SAAM was originally focused on comparing modifiability of different software architectures in an organization's domain. It has since then evolved to a structured method for scenario-based software architecture evaluation. Several quality attributes can be addressed, depending on the type of scenarios that are created during the evaluation process. Case-studies where maintainability and usability are evaluated have been reported in [18], and modifiability, performance, reliability, and security are explicitly stated in [21].
The method consists of five steps. It starts with the documentation of the architecture in a way that all participants of the evaluation can understand. Scenarios are then developed that describe the intended use of the system. The scenarios should represent all stakeholders that will use the system. The scenarios are then evaluated and a set of scenarios that represents the aspect that we want to evaluate is selected. Interacting scenarios are then identified as a measure of the modularity of the architecture. The scenarios are then ordered according to priority, and their expected impact on the architecture. SAAM has been used and validated in several studies [10, 12, 18, 19, 25]. There also exist methods that are extensions and/or further evolutions of SAAM, which are surveyed by Dobrica and Niemelä [11].

ATAM — ARCHITECTURE TRADE-OFF ANALYSIS METHOD

Architecture Trade-off Analysis Method (ATAM) [21] is a scenario-based software architecture evaluation method. The goals of the method are to evaluate an architecture- level design that considers multiple quality attributes and to gain insight as to whether the implementation of the architecture will meet its requirements. ATAM builds on SAAM and extends it to handle trade-offs between several quality attributes. The architecture evaluation is performed in six steps. The first one is to collect scenarios that operationalize the requirements for the system (both functional and quality requirements). The second step is to gather information regarding the constraints and environment of the system. This information is used to validate that the scenarios are relevant for the system. The third step is to describe the architecture using views that are relevant for the quality attributes that were identified in step one. Step four is to analyze the architecture with respect to the quality attributes. The quality attributes are evaluated one at a time. Step five is to identify sensitive points in the architecture, i.e., identifying those points that are affected by variations of the quality attributes. The sixth and final step is to identify and evaluate trade-off points, i.e., variation points that are common to two or more quality attributes. ATAM has been used and validated in several studies [21, 32].

ALMA — ARCHITECTURE-LEVEL MODIFIABILITY ANALYSIS

Architecture-Level Modifiability Analysis (ALMA) [6, 7] is a scenario-based software architecture evaluation method with the following characteristics: focus on modifiability, distinguish multiple analysis goals, make important assumptions explicit, and provide repeatable techniques for performing the steps. The goal of ALMA is to provide a structured approach for evaluating three aspects of the maintainability of software architectures, i.e., maintenance prediction, risk assessment, and software architecture comparison. ALMA is an evaluation method that follows SAAM in its organization. The method specifies five steps: 1. Determine the goal of the evaluation, 2. Describe the software architecture, 3. Elicit a relevant set of scenarios, 4. Evaluate the scenarios, and 5. Interpretation of the results and draw conclusions from them. The method provides more detailed descriptions of the steps involved in the process than SAAM does, and tries to make it easier to repeat evaluations and compare different architectures. It makes use of structural metrics and base the evaluation of the scenarios on quantification of the architecture. The method has been used and validated by the authors in several studies [6, 7, 24].

RARE/ARCADE

RARE and ARCADE are part of a toolset called SEPA (Software Engineering Process Activities) [4]. RARE (Reference Architecture Representation Environment) is used to specify the software architecture and ARCADE is used for simulation-based evaluation of it. The goal is to enable automatic simulation and interpretation of a software architecture that has been specified using the RARE environment. An architecture description is created using the RARE environment. The architecture descriptions together with descriptions of usage scenarios are used as input to the ARCADE tool. ARCADE then interprets the description and generates a simulation model. The simulation is driven by the usage scenarios. RARE is able to perform static analysis of the architecture, e.g., coupling. ARCADE makes it possible to evaluate dynamic attributes such as performance and reliability of the architecture. The RARE and ARCADE tools are tightly integrated to simplify an iterative refinement of the software architecture. The method has, as far as we know, only been used by the authors.

ARGUS-I

Argus-I [37] is a specification-based evaluation method. Argus-I makes it possible to evaluate a number of aspects of an architecture design. It is able to perform structural analysis, static behavioral analysis, and dynamic behavioral analysis, of components. It is also possible to perform dependence analysis, interface mismatch, model checking, and simulation of architecture. Argus-I uses a formal description of a software architecture and its components together with state charts that describe the behavior of each component. The described architecture can then be evaluated with respect to performance, dependence, and correctness. There is no explicit process defined that the evaluation should follow, but some guidance is provided. The evaluation results in a quantification of the qualities of the architecture. The performance of the architecture is estimated based on the number of times that components are invoked. The simulation can be visualized using logs collected during the simulation. The method has, as far as we know, only been used by the authors.

LQN — LAYERED QUEUING NETWORKS

Layered queuing network models are very general and can be used to evaluate many types of systems. Several authors have proposed the use of queuing network models for software performance evaluation [14, 15, 22, 30, 33]. Further, there also exist many tools and toolkits for developing and evaluating queuing network models, e.g., [14, 15]. A queuing network model can be solved analytically, but is usually solved using simulation. The method relies on the transformation of the architecture into a layered queuing network model. The model describes the interactions between components in the architecture and the processing times required for each interaction. The creation of the models requires detailed knowledge of the interaction of the components, together with behavioral information, e.g., execution times or resource requirements. The execution times can either be identified by, e.g. Measurements, or estimated. The more detailed the model is the more accurate the simulation result will be.
The goal when using a queuing network model is often to evaluate the performance of software architecture or a software system. Important measures are usually response times, throughput, resource utilization, and bottleneck identification. In addition, some tools not only produce measures, but also have the ability to visualize the system behavior.

SAM

SAM [38] is a formal systematic methodology for software architecture specification and analysis. SAM is mainly targeted for analyzing the correctness and performance of a system. SAM has two major goals. The first goal is the ability to precisely define software architectures and their properties, and then perform formal analysis of them using formal methods. Further, SAM also supports an executable software architecture specification using time Petri nets and temporal logic. The second goal is to facilitate scalable software architecture specification and analysis, using hierarchical architectural decomposition. The authors have as far as we know, only used the method.

EBAE — EMPIRICALLY-BASED ARCHITECTURE EVALUATION

Lindvall et al. describe in [26] a case study of a redesign/ reimplementation of a software system developed more or less in-house. The main goal was to evaluate the maintainability of the new system as compared to the previous version of the system. The paper outlines a process for empirically based software architecture evaluation. The paper defines and uses a number of architectural metrics that are used to evaluate and compare the architectures. The basic steps in the process are: select a perspective for the evaluation, define/select metrics, collect metrics, and evaluate/compare the architectures. In this study the evaluation perspective was to evaluate the maintainability, and the metrics were structure, size, and coupling. The evaluations were done in a late development stage, i.e., when the systems already were implemented. The software architecture was reverse engineered using source code metrics.

ABAS — ATTRIBUTE-BASED ARCHITECTURAL STYLES

Attribute-Based Architectural Styles (ABASs) [23] build on the concept of architectural styles [9, 35], and extend it by associating a reasoning framework with an architectural style. The method can be used to evaluate various quality attributes, e.g., performance or maintainability, and is thus not targeted at a specific set of quality attribute. The reasoning framework for an architectural style can be qualitative or quantitative, and are based on models for specific quality attributes. Thus, ABASs enable analysis of different quality aspects of software architectures based on ABASs. The method is general and several quality attributes can be analyzed concurrently, given that quality models are provided for the relevant quality attributes. One strength of ABASs is that they can be used also for architectural design. Further, ABASs have been used as part of evaluations using ATAM [21].

SPE — SOFTWARE PERFORMANCE ENGINEERING

Software performance engineering (SPE) [36, 39] is a general method for building performance into software system. A key concept is that the performance shall be taken into consideration during the whole development process, not only evaluated or optimized when the system already is developed. SPE relies on two different models of the software system, i.e., a software execution model and a system execution model. The software execution model models the software components, their interaction, and the execution flow. In addition, key resource requirements for each component can also be included, e.g., execution time, memory requirements, and I/O operations. The software execution model predicts the performance without taken contention of hardware resources into account.

The system execution model is a model of the underlying hardware. Examples of hardware resources that can be modeled are processors, I/O devices, and memory. Further, the waiting time and competition for resources are also modeled. The software execution model generates input parameters to the system execution model. The system execution model can be solved by using either mathematical methods or simulations.

The method can be used to evaluate various performance measures, e.g., response times, throughput, resource utilization, and bottleneck identification. The methods are primarily targeted for performance evaluation. However, the authors argue that their method can be used to evaluate other quality attributes in a qualitative way as well [39].

SUMMARY OF ARCHITECTURE EVALUATION METHODS

Table 2 summarizes the most important characteristics (see Table 1) of our survey of software architecture evaluation methods. As we can see, most of the methods address only one quality attribute of those that we consider in this survey, and the most common attribute to address is performance. Surprisingly, no method was found that specifically address portability or testability. Further, we can observe that only one method exists that support trade-off analysis of software architectures. Finally, we also observe that only two methods seem to have been used by others than the method inventor.

DISCUSSION

Despite the promising number of primary studies found, i.e., 76, it turned out that only 10 software architecture evaluation methods were possible to identify that addressed one or more of the performance, maintainability, testability, or portability quality attributes. There exist several reasons for this large reduction of the number of articles. First, there were some duplicate entries of the same article since we searched several databases. Second, a large portion of the papers evaluated one or several quality attributes in a rather ad hoc fashion. As a result, we excluded those papers from our survey since they did not document a repeatable evaluation method or process. Third, several papers addressed both hardware and software evaluations, thus they did not qualify in our survey with its focus on methods for software architecture evaluation.

Table-2: Summary of evaluation method characteristics.

Name	Category	Quality attributes	Method usage
SAAM [19]	Scenario-based	General	creator [18, 19], other [10, 12, 25]
ATAM [21]	Scenario-based	General, trade-off	creator [21], other [32]
ALMA [6, 7]	Scenario-based	Modifiabity	creator [6, 7, 24]
RARE/ARCADE [4]	Simulation-based	Performance	creator [4]
ARGUS-I [37]	Simulation-based	Performance	creator [37]
LQN [33]	Simulation-based	Performance	creator [1, 33]
SAM [38]	Simulation-based	Performance	creator [38]
EBAE [26]	Experience-based, metrics	Maintainability	creator [26]
ABAS [23]	Experience-based	General	creator [23]
LQN [33]	Simulation-based, mathematical modelling	Performance	creator [33, 1], other
SPE [36, 39]	Simulation-based, mathematical modelling	Performance	creator [36, 39]

Continuing with the ten remaining articles, we found that five of the methods addressed only one single quality attribute. Only one (ATAM) of the remaining five methods addressing multiple attributes provides support for trade-off analysis between the quality attributes. No specific methods evaluated testability or portability explicitly. These quality attributes could be addressed by any of the three evaluation methods that are more general in their nature, i.e., that could address more arbitrary selected quality attributes, ATAM [21], SAAM [19], or the method by Lindvall et al. [26].

Many of the methods have been used several times of the authors. Multiple use of the method indicates an increase in validity of the method. However, only two methods have been used by others than the original authors of the method. We believe that external use of a method is an indication of the maturity of the method. These two methods are SAAM and ATAM. However, experience papers that use a method in whole or part are particularly difficult to identify, since the evaluation method that has been used is not always clearly stated.

CONCLUSIONS

The architecture of a software system has been identified as an important aspect in software development, since the software architecture impacts the quality attributes of a system, e.g., performance and maintainability. A good software architecture increases the probability that the system will fulfill its quality requirements. Therefore, methods for evaluating the quality attributes of software architectures are important.

In this paper, we present a survey of evaluation methods for software architecture quality attribute evaluation. We focus on methods for evaluating one or several of the quality attributes performance, maintainability, testability, and portability. Methods that evaluate several quality attributes and/or trade-off analysis are especially interesting. Based on a broad literature search in major scientific publication databases, e.g., Inspec, and reviewing of 76 articles, we present and compare ten evaluation methods. We have found that many evaluation methods only address one quality attribute, and very few can evaluate several quality attributes simultaneously in the same framework or method. Specifically, only one of the methods includes trade-off analysis. Further, we have identified that many methods are only used and validated by the method inventors themselves.

In summary, our results suggest

•An increased research focus on software architecture evaluation methods than can address several quality attributes simultaneously,

•An increased research focus on software architecture evaluation methods than can address the possible tradeoffs between different quality attributes, and

•An increased focus on validation of software architecture evaluation methods by people other than the method inventors.

REFERENCES

[1] Aquilani, F., Balsamo, S., and Inverardi, P., "Performance Analysis at the Software Architectural Design Level," Performance Evaluation, vol. 45, pp. 147-178, 2001.

[2] Avritzer, A. and Weyuker E. J., "Metrics to Assess the Likelihood of Project Success Based on Architecture Reviews," Empirical Software Engineering, 4(3):199-215, 1999.

[3] Babar, M. A., Zhu, L., and Jeffery, R., "A framework for classifying and comparing software architecture evaluation methods," Proc. Australian Software Engineering Conference, pp. 309-318, 2004.

[4] Barber, K. S., Graser, T., and Holt, J., "Enabling iterative software architecture derivation using early non-functional property evaluation," Proc. 17th IEEE International Conference on Automated Software Engineering, pp. 23-27, 2002.

[5] Bass, L., Clements, P., and Kazman, R., Software Architecture in Practice, ISBN 0-631-21304-X, Addison-Wesley, 2003.

[6] Bengtsson, PO., Architecture-Level Modifiability Analysis, ISBN 91-7295-007-2, Blekinge Institute of Technology, Dissertation Series No 2002-2, 2002.

[7] Bengtsson, PO., Lassing, N., and Bosch, J., "Architecture Level Modifiability Analysis (ALMA)," Journal of Systems and Software, vol. 69, pp. 129-147, 2004.

[8] Bosch, J., Design & Use of Software Architectures – Adopting and evolving a product-line approach, ISBN 0-201- 67494-7, Pearson Education, 2000.

[9] Buschmann, F., Meunier, R., Rohnert, H., Sommerland, P , and Stal, M., Pattern-Oriented Software Architecture – A System of Patterns, ISBN 0-471-95869-7, Wiley, 1996.

[10] Castaldi, M., Inverardi, P., and Afsharian, S., "A case study in performance, modifiability and extensibility analysis of a telecommunication system software architecture," Proc. 10th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, pp. 281-290, 2002.

[11] Dobrica, L. and Niemelä, E., "A Survey On Architecture Analysis Methods," IEEE Transactions on Software Engineering, 28(7):638-653, 2002.

[12] Eikelmann, N. S. and Richardson, D. J., "An Evaluation of Software Test Environment Architectures," Proc. 18th International Conference on Software Engineering, pp. 353-364, 1996.

[13] Etxeberria, L. and Sagardui, G., "Product-Line Architecture: New Issues for Evaluation," Lecture Notes in Computer Science, Volume 3714, ISBN 3-540-28936-4, Springer-Verlag GmbH, 2005.

[14] Franks, G., Hubbard, A., Majumdar, S., Petriu, D., Rolia, J., and Woodside C.M., "A Toolset for Performance Engineering and Software Design of Client-Server Systems," Performance Evaluation, 24(1-2):117-136, November 1995.

[15] Gunther, N., The Practical Performance Analyst, ISBN 0- 07-912946-3, McGraw-Hill, 1998.

[16] Holzmann, G.J., "The Model Checker SPIN," IEEE Transactions on Software Engineering, 23(5):279-295, May 1997.

[17] IEEE std 610.12-1990 (n.d.). IEEE Standard Glossary of Software Engineering Terminology, 1990. Retrieved January 19, 2006. Web site: http://ieeexplore.ieee.org/

[18] Kazman, R., Abowd, G., Bass, L., and Clements, P., "Scenario-based analysis of software architecture," IEEE Software, 13(6):47-55, November 1996.

[19] Kazman, R., Bass, L., Abowd, G., and Webb, M., "SAAM: A Method for Analyzing the Properties of Software Architectures," Proc. 16th International Conference of Software Engineering, pp. 81-90, 1994.

[20] Kazman, R., Bass, L., Klein, M., Lattanze, T., and Northrop, L., "A Basis for Analyzing Software Architecture Analysis Methods," Software Quality Journal, 13(4):329-355, 2005.

[21] Kazman, R., Klein, M., Barbacci, M., Longstaff, T., Lipson, H., and Carriere, S. J., "The Architecture Tradeoff Analysis Method," Proc. 4th IEEE International Conference on Engineering of Complex Computer Systems, pp. 68-78, 1998.

[22] King, P., Computer and Communication Systems Performance Modelling, ISBN 0-13-163065-2, Prentice Hall, 1990.

[23] Klein, M. and Kazman, R., "Attribute-Based Architectural Styles," CMU/SEI-99-TR-22, Software Engineering Institute, Carnegie Mellon University, 1999.

[24] Lassing, N., Bengtsson, P., Van Vliet, H., and Bosch, J., "Experiences with ALMA: Architecture-Level Modifiability Analysis," Journal of Systems and Software, 61(1):47-57, March 2002.

[25] Lassing, N., Rijsenbrij, D., and van Vliet, H., "Towards a Broader View on Software Architecture Analysis of Flexibility," Proc. Sixth Asia-Pacific Software Engineering Conference, pp. 238-245, 1999.

[26] Lindvall, M., Tvedt, R. T., and Costa, P., "An empiricallybased process for software architecture evaluation," Empirical Software Engineering, 8(1):83-108, 2003.

[27] Liu, A. and Gorton, I., "Accelerating COTS Middleware Acquisition: The i-Mate Process," IEEE Software, 20(2): 72- 79, March/April 2003.

[28] Luckham, D. C., "Rapide: A Language and Toolset for Simulation of Distributed Systems by Partial Orderings of Events," Proc. DIMACS workshop on Partial order methods in verification, pp. 329-357, Princeton, 1997.

[29] Luckham, D., John, K., Augustin, L., Vera, J., Bryan, D., and Mann, W., "Specification and Analysis of System Architecture using RAPIDE," IEEE Transactions on Software Engineering, 21(4):336-335, 1995.

[30] Menascé, D., Almeida, V., and Dowdy, L., Capacity Planning and Performance Modelling, ISBN 0-13-035494-5, Prentice Hall, 1994.

[31] Mikk, E., Lakhnech, Y., Siegel, M., and Holzmann, G.J., "Implementing Statecharts in PROMELA/SPIN," Proc. 2nd IEEE Workshop on Industrial Strength Formal Specification Techniques, pp. 90-101, October 1998.

[32] Mukkamalla R., Britton M., and Sundaram P., "Scenario- Based Specification and Evaluation of Architectures for Health Monitoring of Aerospace Structures," Proc. 21st Digital Avionics Systems Conference, Vol 2, pp. 12E1-1-12E1- 12, October 2002.

[33] Petriu, D., Shousha, C., and Jalnapurkar, A., "Architecture- Based Performance Analysis Applied to a Telecommunication System," IEEE Transactions on Software Engineering, 26(11):1049-1065, November 2000.

[34] Reusner, R., Schmidt, H.W., and Poernomo, I. H., "Reliability prediction for component-based software architectures," Journal of Systems and Software, 66(3):76-252, 2003.

[35] Shaw, M. and Garlan, D., Software Architecture: Perspectives on an Emerging Discipline, ISBN 0-13-182957-2, Prentice-Hall, 1996.

[36] Smith, C. and Williams, L., Performance Solutions, ISBN 0- 201-72229-1, Addison-Wesley, 2002.

[37] Vieira, M. E. R., Dias, M. S., and Richardson, D. J., "Analyzing software architectures with Argus-I," Proc. 22nd International Conference on Software Engineering, pp. 758- 761, 2000.

[38] Wang, J., He, X., and Deng, Y., "Introducing Software Architecture Specification and Analysis in SAM Through an

Example," Information and Software Technology, 41(7):451-467, May 1999.

[39] Williams, L. G. and Smith, C. U., "Performance Evaluation of Software Architectures," Proc. 1st International Workshop on Software and Performance, pp. 164-177, 1998.