International Journal of Computer Applications (0975 – 8887)
Volume 49– No.16, July 2012

Software Architecture Evaluation Methods – A survey

P. Shanmugapriya,
Research Scholar,
Department of CSE,
SCSVMV University,
Enathur, Tamilnadu,INDIA

R. M. Suresh
Principal,
Jerusalem College of Engineering,
Chennai, Tamilnadu,INDIA

ABSTRACT
Software architectural evaluation becomes a familiar practice in software engineering community for developing quality software. Architectural evaluation reduces software development effort and costs, and enhances the quality of the software by verifying the addressability of quality requirements and identifying potential risks and it provides assurance to developers that their chosen architecture will meet both functional and non-functional quality requirements. This paper presents a discussion on different software architectural evaluation methods and techniques and concentrates on summarizing the importance of the different early and late evaluation methods, similarities and difference between them, their applicability, strengths and weaknesses.

Keywords
Software architectural, evaluation, early and late evaluation methods

1. INTRODUCTION
Software architecture evaluation is a technique or method which determines the properties, strengths and weaknesses of software architecture or software architectural style or a design pattern. Software architectural evaluation provides assurance to developers that their chosen architecture will meet both functional and non-functional quality requirements. An architectural evaluation should provide more benefits than the cost of conducting the evaluation itself [1]. Software architectural evaluation ensures increased understanding and documentation of the system, detection of problems with existing architecture, and enhanced organizational learning. Several methods and techniques have been proposed for software architectural evaluation. Among those *scenario-based approaches* are considered quite mature [17, 6]. There are also *attribute model-based methods* [25] and *quantitative models* [39] for software architecture evaluation. Other approaches have been developed to systematically justify the properties of architectural styles [25] and design patterns [33, 20]. The goal of this paper is to review existing software architectural evaluation methods and to classify the methods in the form of taxonomy. The presented taxonomy also considers two phases of a software life cycle: early and late.

2.EVALUATION METHODS
A number of evaluation methods have been developed which are applicable in different phases of the software development cycle. The main two opportunities for evaluation are before and after implementation [15]. Early software architecture evaluation methods are applied to software architecture before its implementation. Quality goals can primarily be achieved if the software architecture is evaluated with respect to its specific quality requirements at the early stage of software development. Late software architecture evaluation methods identify the difference between the actual and planned architectures. These methods provide useful guidelines of how to reconstruct the actual architecture, so that it conforms to the planned architecture.

3.EARLY EVALUATION METHODS
Early software architectural evaluation can be conducted on the basis of the specification and description of the software architecture. The scenario-based approaches are flexible and simple [5, 6]. Mathematical model-based evaluation techniques for assessing the operational quality attributes, such as reliability and performance are also well used, particularly in real-time software systems.

3.1Scenario-based Software Architecture Evaluation Methods
Scenario-based evaluation methods evaluate software architecture's ability with respect to a set of scenarios of interest. Scenario is brief descriptions of a single interaction of a stakeholder with a system [8]. Different scenario based methods have been developed so far [22, 23, 29, 31, 11, 38, 40, 44].
The scenario-based evaluation methods offer a systematic means to investigate software, architecture using scenarios. These methods determine whether software architecture can execute a scenario or not. Evaluation team explores/maps the scenario onto the software architecture to find out the desired architectural components and their interactions, which can accomplish the tasks expressed through the scenario. If the software architecture fails to execute the scenario, these methods list the changes to the software architecture required to support the scenario and estimate the cost of performing the changes.
Scenario-based evaluation methods require presence of relevant stakeholders to elicit scenarios according to their requirements. Scenario-based methods can ensure discovery of problems in software architectures from different point of views by incorporating multiple stakeholders during the scenario elicitation process whereas the end user can indicate the performance issues. The following are the set of Scenario-based evaluation methods

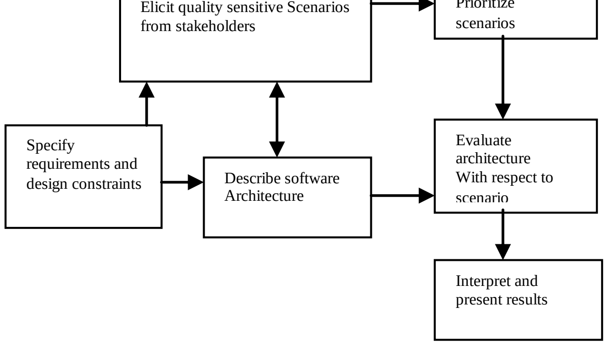
Elicit quality sensitive Scenarios from stakeholders
Prioritize scenarios
Specify requirements and design constraints
Describe software Architecture
Evaluate architecture With respect to scenario
Interpret and present results

Fig – 1: Common behavior in scenario-based evaluation methods

SAAM (Scenario-based Software Architecture Analysis Method) [22, 46, 48]
ATAM (Architecture based Tradeoff Analysis Method) [46, 47]
ALPSM (Architecture-Level Prediction of Software Maintenance) [10] and ALMA (Architecture-Level Modifiability Analysis) [11]
CBAM (Cost-Benefit Analysis Method)[46,49]
FAAM (Family-Architecture Assessment Method) [50]
SALUTA (Scenario-based Architecture Level UsabiliTy Analysis) [19]
SBAR (Scenario-Based Architecture Reengineering) [9]
SAAMCS (SAAM for Complex Scenarios) [29]
ESAAMI (Extending SAAM by Integration in the Domain) [31]
ASAAM (Aspectual Software Architecture Analysis Method) [40]
SACAM (Software Architecture Comparison Analysis Method)[12] and DoSAM (Domain Specific Software Architecture Comparison Model)[13]

Table 1. Comparison of the various scenario-based evaluation methods

Evaluation Method	**Main objective**	**Steps in Evaluation**	**Scenario classification & impact analysis**	**Approaches used**	**Objects analyzed**	**Addressed QAs**
SAAM	Architectural suitability and risks analysis	Six activities, some activities carried out in parallel- includes no preparation activities	Direct and indirect scenarios. Counts the number of components affected by the scenarios	Scenario elicitation via brainstorming with Stakeholders. Mapping scenarios onto SAs to verify functionality or estimate change cost	Architectural documentation, especially, showing the logical views	Mainly modifiability but can be adapted for others
ATAM	Sensitivity and tradeoff analysis	Nine activities, some activities carried out in parallel; includes preparation activities	Use-case, growth and Exploratory scenarios. Counts the sensitivity points and Tradeoff points	Creation of utility tree to elicit scenarios. Analysis of architectural approaches using analytic models to identify tradeoff points and risks	Architectural approaches or styles; architectural documentation mainly showing Kurcten's 4+1	Multiple QAs
ALMA	Maintenance cost prediction, risk assessment, architectures comparison	Five, the scenario elicitation activity consists of six activities, executed sequentially; no preparation activities	Change Scenarios. Estimates the change of the size of each component	Scenario elicitation based on the goals of the evaluation. Mapping scenarios onto SAs to estimate maintenance cost considering ripple effects.	Architectural documentation, concerning the system's structure comprising components and connectors (like the logical view)	Reusability
CBAM	Provide business measures for particular system changes Make explicit the uncertainty associated with the estimates	Six main steps, quantify the Quality benefits, cost and schedule implications of the architectural strategies	Direct, indirect and exploratory scenarios. Counts the Time and cost utilized by the scenarios	Analyze the benefits of the different architectural strategies ,Assess the quality, and calculate the desirability with respect to cost and time factor	Time and Costs factors involved in analyzing the quality factors and architectural documentation	Costs, Benefits, and Schedule Implications
FAAM	Emphasis on empowering the teams in applying the	Six main steps, these steps must be adapted in	Focusing on interoperable scenarios. The general assessment	Creation of guidelines and templates in generating	It has a well-defined process workbench Description .	Interoperabilit y and Extensibility

	FAAM session	response to general architecture assessment experience of the organization.	process is tailored for the domain of information-systems families.	change-case-guidelines and templates, requirements-ranking criteria, family-feature maps, migration-maps, family-context diagrams.	Architectural documentation, especially, showing the logical views	
SALUTA	Usability analysis	Four, executed sequentially-no preparation activities	Usage scenarios. Qualitative analysis	Usage profiles to articulate Usability requirements. Scenario walkthrough to analyze the extracted usability properties and patterns.	Usability patterns, usability properties ; No particular view is recommended.	Usability
SBAR	SA reengineerin g to achieve QAs	Three, executed repeatedly; no preparation activities	Development and operational Scenarios. Qualitative analysis	Quality assessment using one of the four techniques and architecture transformation	Initially created architectural documentation. No particular view is considered	Multiple QAs
SAAMCS	Developing complex scenarios to achieve domain specific flexibility	Three, two executed in parallel; no preparation activities	Complex Scenarios. Same as SAAM but defines four level of impacts	Analysis of SAs to determining the values of the three factors that make scenarios complex to implement.	Micro - architectural and macro architectural documentation	Flexibility
ESAAMI	Integrating SAAM in a domain specific reuse based development process	Same as SAAM but considers the existence of reusable knowledge base	Same as SAAM.	Formulation of an analysis template to collect reusable products	Reusable software architecture documentation	Modifiability
ASAAM	Architectural aspect analysis	Same as SAAM but includes the architectural aspects and tangled components identification.	Direct, indirect and aspectual scenarios. Qualitative analysis	Architectural aspect identification from direct and indirect scenarios. Aspectual scenarios interactions to identify different components, such as tangled and ill-defined components	Same as SAAM	Modifiability
SACAM	Comparing software architectures from different domains	Six activities, one executes repeatedly; includes preparation activities	Same as ATAM. Determines metrics	Collating comparison criteria presenting candidate SAs at a common architectural view, and analyzing fitness of the SAs w.r.t. to the criteria	Same as ATAM	Multiple QAs
DoSAM	Comparing software	Six activities, executed	Not performed. Same as	Creation of a DACF,	Architectural documentation.	Multiple QAs

	architectures from a specific domain	sequentially; no preparation activities	SACAM	candidate architectures mapping to the DACF, assessment of QAs employing metrics and comparing architectures based on the metrics values of the QAs	No particular view is recommended.	

3.2Mathematical Model based Software Architecture Evaluation

Most scenario-based software architecture evaluation methods (with the exception of ATAM and SBAR) use qualitative reasoning for assessing development-time quality attributes. However, to measure the fitness of the safety-critical software systems, such as medical, aircraft, and space mission, it is also important to quantitatively assess operational quality attributes. Therefore, a number of mathematical model- based software architecture evaluation methods have been developed. These methods model software architectures using well-known mathematical equations. Then, these methods use the models to obtain architectural statistics, for instance, *mean execution time* of a component. These architectural statistics are used to estimate operational quality attributes. Reliability and performance are two important operational quality attributes. To assess these two quality attributes a wide range of mathematical-models have been developed. Two different approaches are used for assessing reliability of software architecture. They are path based [26, 35, 45] and state based [14, 27, 28]. SPE [36], WS [43], PASA [42], CM [16], BIM [7], ABI [4], AABI [2] are the approaches for predicting performance at the architectural level.

3.3Software Architecture-based Reliability Analysis

According to ANSI [3], a software system's reliability is defined as the probability of the software operating without failure for a specified period of time in a specified environment. Reliability is defined in terms of the mean time between failures or its reciprocal, the failure rate. Software failures may occur for several reasons: errors and ambiguities in architectural design, carelessness or incompetence in writing code, inadequate testing, incorrect or unexpected usage of the software or other unforeseen problems [24,37]. To reduce the probability of software failures, different reliability models have been developed over the past two decades. Early reliability models are based on reliability engineering, particularly hardware reliability. These approaches make use of extensive experience and provide advanced mathematical formalism for building software reliability models. These models complement testing by providing an estimate of a program's ability to operate without failure.

3.4Software architecture - based performance analysis

Software architecture plays an important role in meeting a software system's performance. Performance depends largely on the frequency and nature of inter-component communication and the performance characteristics of the components themselves.

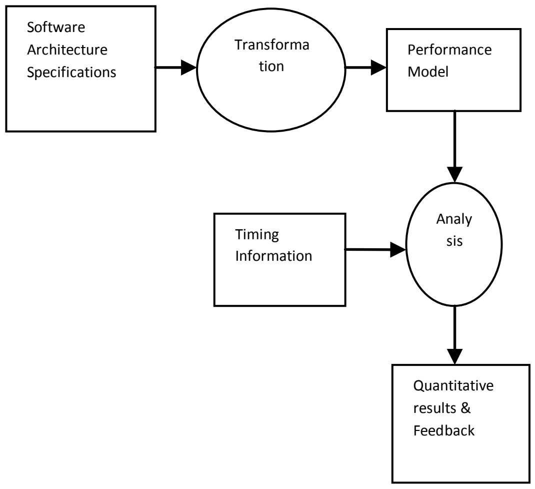
Software Architecture Specifications
Transformation
Performance Model
Timing Information
Analysis
Quantitative results & Feedback

Fig – 2: Software architecture-based performance analysis

Different software architecture-based methodologies have been developed to predict performance attributes, such as throughput, utilization of resources, and end-to-end latency. Architecture-based performance analysis methodologies transform the specification of software architecture into desirable models. Then, timing information is added to these models. After that, they are analyzed to estimate performance attributes quantitatively and to provide feedback about the software architecture. These methodologies work based on availability of software artifacts, such as requirement and architecture specifications and design documents. Since performance is a runtime attribute, these methodologies require suitable description of the dynamic behavior of a software system. Often, automatic tools are used to perform performance analysis once the performance models are created. The general framework for analyzing performance at architectural level is shown in (see Fig-2.) Some of the advantages of architecture-based performance analysis methodologies are as follows [21]: (i) they can help predict the performance of a system early in the software life cycle. (ii)They can be used to guarantee that performance goals are met. (iii)They can also be used to compare the performance of different architectural choices. (iv)They can help in finding bottleneck resources and identifying potential timing problems before the system is built.

4.LATE EVALUATION METHODS

The software systems are continuously modified, to fix problems and adapt to new requirements. The developers who work under intense time pressure and heavy work load cannot always follow the best way to implement changes. As a result the actual architecture may deviate from the planned one. Late software architecture evaluation methods identify the difference between the actual and planned architectures. These methods provide useful guidelines of how to reconstruct the actual architecture, so that it conforms to the planned architecture. During the testing phase, late software architecture evaluation methods are also applied to check the compliance of the source code to the planned design. According to Fiutem and Antoniol [18], the economics of the design-code compliance verification process not only saves time in updating designs, but also improves design artifacts as well as software development and maintenance processes. Late software architecture evaluation can use data measured on the implementation of software architecture. Metrics can be used to reconstruct the actual software architecture, allowing it to be compared to the planned architecture.

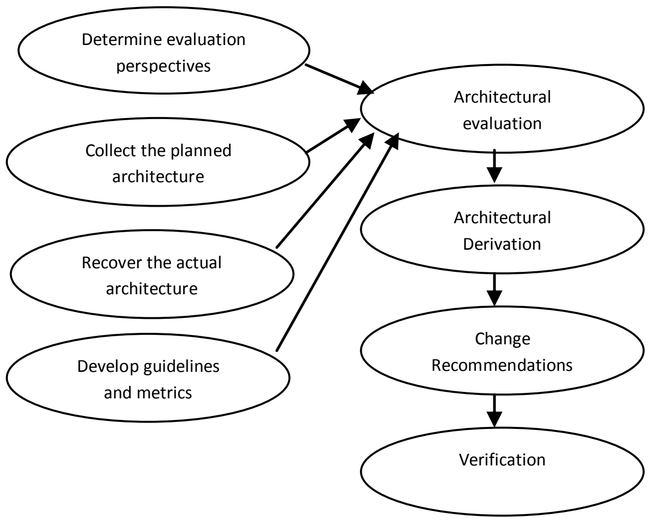
Determine evaluation perspectives
Collect the planned architecture
Recover the actual architecture
Develop guidelines and metrics
Architectural evaluation
Architectural Derivation
Change Recommendations
Verification

Fig – 3: Activities of a late software architecture evaluation

Table 2. Comparison of the various late evaluation methods

Name of Approach	**Main Objective**	**Approach used**	**Steps in Late Evaluation**	**infringement identified**
Tvedt et al.'s Approach[41]	To avoid system degeneration by actively and systematically detecting and correcting deviations of the actual software architecture from the planned architecture.	Checking the functional & non-functional requirements	Identify actual architecture. Check the deviation from planned & actual architecture. Change recommendations will be placed.	Design pattern, misplaced classes and minor violations
Lindvall et al.'s Approach [30]	To identifying the maintainability problems in the software system and the system has been restructured as component-based system using a new design pattern	Designed a component bases system (EMS) to addressed maintainability problems in the software system	Compare the new actual architecture with Previous actual architecture and planned actual architecture.	Inter module coupling violation
Fiutem and Antoniol's Approach (Tool-based) [18]	To determine the inconsistency	Compares the recovered design with the planned design	Compare and determines the inconsistency.	Some Code violation
Murphy et al.'s Approach (Tool-based) [32]	To check the compliance of source code to the planned architecture.	Sequence of reflexion models are used to compare the layer architecture design.	Check whether the high level model agrees or disagrees with the source code. Checks the declarative mapping between the two models	Calls between modules violation
Sefika et al.'s Approach (Tool-based) [34]	To determine design-implementation congruence at various levels of abstraction .It is a hybrid approach that integrates logic based static and dynamic visualizations.	Hybrid Approach	Integrates logic, static and dynamic visualizations	Design pattern violation

5. DISCUSSIONS

There are seven comparison criteria are used to compare twelve scenario based evaluation methods. The following provide a brief description of the comparison criteria. Most scenario- based methods are similar at a coarse-grained level. There are significant differences at a finer-grained level. Different methods classify scenarios differently. Some methods classify scenarios as direct and indirect, while others employ use-case scenarios or growth scenarios. One method might count the number of components affected by a particular scenario, while another might use metrics for each quality attribute. From the comparative analysis, it is understood that

- Some scenario-based methods, particularly SAAM, ATAM and ALMA, have been successfully applied in different industrial settings.
- Scenario-based evaluation methods basically use change scenarios and scenario interactions to expose potential problem areas in the architecture.
- These methods measure the risks of a software system by estimating the degree of changes that software architecture requires to implement a scenario.
- In scenario-based methods, it is hard to assess scenario coverage.
- Developing a framework or methodologies will help to determine the scenario coverage.
- SAAMCS is a good step towards framework.
- Very few scenario-based methods are tool-supported; e.g., only the activities of SAAM and ATAM are even partially supported by tools.
- Performance-based evaluation approaches appear to be more matured than those for reliability.
- Performance-based approaches are mostly tool-supported though none of the tools can support the complete architectural analysis process.
- Performance-based evaluation approaches also consider concurrent and non-deterministic behavior of software components.
- Reliability-based evaluation approaches do not have that much tool-support and these methods are still evolving to support concurrent and non-deterministic behaviors of software components.
- Mathematical model- based evaluation is well suited for component-based software system.
- It is difficult to convert software architecture into a mathematical model and the use of mathematical model-based architectural evaluation methods is comparatively lower than scenario-based evaluation methods.
- In Late software architecture evaluation, its main aim is to provide an inexpensive and quick means for detecting violations to the software architecture with the evolution of software systems. So, these evaluation methods are mostly tool-supported.
- The metrics-based approaches have been only used to evaluate the software architecture with respect to maintainability perspective.
- The late software architecture evaluation greatly addresses the Analysis of design-code consistency.
- The late software architecture evaluation has no formal framework and taxonomy to analyze design-code inconsistencies and prioritize the interventions to make design.

Number of problems was identified from the analysis. One of the most sensitive problem found in scenario based methods are scenario coverage problem. A new knowledge based software architecture evaluation model [51] is developed with Quality Function Deployment (QFD) technique. The difficulties and enhancement opportunities of the architecture review process are investigated, in particular in the context of the knowledge it produces and requires, scenarios covered and proposed a conceptual solution for enhancing the review process and embedding knowledge data usage within it.

6. CONCLUSION

In this paper, the software architectural evaluation methods using taxonomy is presented. The taxonomy shows the architectural evaluation methods and techniques. One of the main problems identified in software architecture evaluation is that it is hard to assess coverage of scenarios. There is no particular number of scenarios, the execution of which guarantees scenario coverage optimally. No method offers systematic methodologies that help elicit such important scenarios. So Scenario coverage needs more attention. From the observation, it is clear that there needs a model to determine the scenario coverage problem and to analyze design-code inconsistencies and prioritize the interventions to make design. A conceptual model is suggested here to overcome the major problems raised in scenario based evaluation methods. The design-code compliance verification process saves time in updating designs, and also improves design artifacts as well as software development and maintenance processes. This process in late evaluation needs to be strengthened more to improve early design.

7. REFERENCES

[1] G. Abowd, L. Bass, P. Clements, Rick Kazman, L. Northrop, and A. Zaremski. Recommended Best Industrial Practice for Software Architecture Evaluation (CMU/SEI-96-TR-025). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 1996.

[2] F. Andolfi, F. Aquilani, S. Balsamo, and P. Inverardi. Deriving QNM from MSCs for Performance Evaluation of SA. In *the Proceedings on 2nd International Workshop on Software and Performance*, pp. 2000

[3] ANSI/IEEE, "Standard Glossary of Software Engineering Terminology", STD-729-1991, ANSI/IEEE, 1991

[4] F. Aquilani, S. Balsamo, P. Inverardi.*Performance Analysis at the software architecture design level*. Technial Report TRSAL- 32, Technical Report SaladinProject.

[5] M. A. Babar, L. Zhu and R. Jeffery. A Framework for Classifying and Comparing Software Architecture Evaluation Methods. In *the Proceedings on Australian Software engineering*, pp. 309-318, 2004.

[6] M. A. Babar and I. Gorton. Comparison of Scenario-Based Software Architecture Evaluation Methods. In *the Proceedings on Asia-Pacific Software Engineering Conference*, pp. 584-585, 2004.

[7] S. Balsamo, P. Inverardi and C. Mangano. An approach to performance evaluation of software architectures. In *the Proceedings on 2nd International Workshop on Software and Performance*, pp. 178-190, 1998

[8] L. Bass, P. Clements and R. K. Kazman. Software Architecture in Practice. SEI Series in Software

Engineering. Addison-Wesley, 1998. ISBN 0-201-19930-0.

[9] P. Bengtsson and J. Bosch. Scenario Based Software Architecture Reengineering. In *the Proceedings of International Conference of Software Reuse*, pp. 308-317, 1998.

[10] P. Bengtsson, J. Bosch. Architecture Level Prediction of Software Maintenance. In *the Proceedings on 3rd European Conference on Software Maintenance and Reengineering*, pp. 139-147, 1999.

[11] P. Bengtsson, N. Lassing, J. Bosch, and H. V. Vliet. Architecture-Level Modifiability Analysis. Journal of Systems and Software, vol. 69, 2004.

[12] J. K. Bergey, M. J. Fisher and L. G. Jones and R. Kazman. Software ArchitectureEvaluation with ATAMSM in the DoD System Acquisition Context. CMU/SEI-99-TN-012. Pittsburg, PA: Software Engieering Institute, Carnegie Mellon University, 1999.

[13] K. Bergner, A. Rausch, M. Sihling and T. Ternit. DoSAM - Domain-Specific Software Architecture Comparison Model. *In the Proceedings of the International Conference on Quality of Software Architectures*, pp. 4-20, 2005.

[14] R. C. Cheung. *A user-oriented software reliability model*. IEEE Trans. on Software Engineering, vol. 6, pp. 118-125, 1980.

[15] P. Clements and R. K. Kazman, M. Klein. Evaluating Software Architectures: Methods and Case Studies. Addison-Wesley Professional; 2002. ISBN 0-201-70482X

[16] V. Cortellessa and R. Mirandola. Deriving a Queueing Network based Performance Model from UML Diagrams. In *the Proceedings on 2nd International Workshop on Software and Performance*, pp. 58-70, 2000.

[17] L. Dobrica and E. Niemela. A Survey on Software Architecture Analysis Methods. *IEEE Transactions on Software Engineering*, vol. 28, no. 7, pp. 638-653, July 2002.

[18] R. Fiutem , and G. Antoniol. Identifying design-code inconsistencies in object- oriented software: a case study. In *the Proceedings of the International Conference on Software Maintenance*, pp. 94-102, 1998

[19] E. Folmer, J. Gurp and J. Bosch. Software Architecture Analysis of Usability. In *the Proceedings on 9th IFIP Working Conference on Engineering Human Computer Interaction and Interactive Systems*, pp. 321-339, 2004.

[20] E. Golden, B.E. John and L. Bass. The value of a usability-supporting architectural pattern in software architecture design: a controlled experiment. In *the Proceedings on 27th international conference on Software engineering*, pp. 460- 469, 2005.

[21] T. Kauppi. *Performance analysis at the software architectural level*. Technical report, ISSN: 14550849, 2003.

[22] R. Kazman, G. Abowd, and M. Webb. SAAM: A Method for Analyzing the Properties of Software Architectures. *In the Proceedings on 16th International Conference on Software Engineering*, pp. 81-90, 1994.

[23] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, and J. Carriere. The Architecture Tradeoff Analysis Method. In *the Proceedings on ICECCS*, pp.68-78, 1998.

[24] P. A. Keiller, and D. R. Miller. On the Use and the Performance of Software Reliability Growth Models. *Software Reliability and Safety*, Elsvier, pp. 95-117, 1991.

[25] M. H. Klein, R. Kazman, L. Bass, J. Carriere, M. Barbacci and H. Lipson. Attribute-Based Architectural Styles. In *the Proceedings on First Working IFIP Conference on Software Architecture*, pp. 225-243, 1999.

[26] S. Krishnamurthy and A. P. Mathur. On the estimation of reliability of a software system using reliabilities of its components. In *the Proceedings of 8th Int'l Symp. Software Reliability Engineering*, pp. 146-155, 1997.

[27] P. Kubat. Assessing reliability of modular software. Operation Research Letters, 8:35-41, 1989.

[28] J. C. Laprie. Dependability evaluation of software systems inoperation. IEEE Trans. on Software Engineering, vol. 10(6), pp. 701-714, 1984.

[29] N. Lassing, D. Rijsenbrij, and H. v. Vliet. On Software Architecture Analysis of Flexibility, Complexity of Changes: Size isn't Everything. In *the Proceedings of 2nd Nordic Software Architecture Workshop*, 1999.

[30] M. Lindvall, R. T. Tvedt and P. Costa. An empirically-based process for software architecture evaluation. Empirical Software Engineering 8(1): 83Y108, 2003.

[31] G. Molter. Integrating SAAM in Domain-Centric and Reuse-based Development Processes. In *Proceedings of the 2nd Nordic Workshop on Software Architecture*, 1999.

[32] G. C. Murphy, D. Notkin, and K. Sullivan. Software re°exion models: bridging the gap between source and high-level models. In *the Proceedings of the 3rd ACM SIGSOFT symposium on Foundations of software engineering*, pp. 18 - 28, 1995.

[33] L. Prechelt, B. Unger, W. F. Tichy, P. Brssler and L. G. Votta. A Controlled Experiment in Maintenance Comparing Design Patterns to Simpler Solutions. IEEE Transactions on Software Engineering, vol. 27, pp. 1134-1144, 2001.

[34] M. Sefika, A.Sane and R. H. Campbell. Monitoring compliance of a software system with its high level design models. In the *Proceedings of the 18th International Conference on Software Engineering (ICSE)*, pp. 387-397, 1993.

[35] M. Shooman. Structural models for software reliability prediction. In *the Proceedings of 2nd International Conference on Software Engineering*, pp. 268-280, 1976.

[36] C. U. Smith. Performance Engineering of Software Systems. Addison- Wesley, Massachusetts, 570 p., 1990.

[37]SoftwareReliability.http://www.ece.cmu.edu/~koopman/des_s99/sw_reliability/#reference

[38]C.Stoermer,F.Bachmann, C. Verhoef, SACAM: The Software Architecture Comparison Analysis Method, Technical Report, CMU/SEI-2003-TR-006, 2003.

[39] M. Svahnberg, C. Wohlin, L. Lundberg, and M. Mattsson. A Method for Understanding Quality Attributes in Software Architecture Structures. In *Proceedings of the* 14*th International Conference on Software Engineering and Knowledge Engineering*, 2002.

[40] B. Tekinerdogan. ASAAM: aspectual software architecture analysis method. In the Proceedings of the Working IEEE/IFIP Conference on Software Architecture (WICSA'04), June 2004, pp. 5-14.

[41] R.T. Tvedt, M. Lindvall, and P. Costa. A Process for Software Architecture Evaluation using Metrics. In *the proceedings of* 27*th Annual NASA Goddard/IEEE*, pp. 191-196, 2002.

[42] L.G. Williams and C.U. Smith. PASA: A method for the Performance Assessment of Software Architectures. In *the Proceedings of the Third International Workshop on Software and Performance (WOSP '02)*, pp. 179-189, 1990,.

[43] L.G. Williams and C.U. Smith. Performance Engineering of Software Architectures. In *the Proceeding on Workshop Software and Performance*, pp. 164 - 177, 1998.

[44] S. M. Yacoub, and H. Ammar. A methodology for architectural-level reliability risk analysis. IEEE Transactions on Software Engineering 28: 529-547,2002.

[45] S. Yacoub, B. Cukic, and H. Ammar. Scenario-based reliability analysis of component-based software. In *the Proceedings of 10th Int'l Symp. Software Reliability Engineering*, pp. 22-31, 1999.

[46] Paul Clements, Rick Kazman and Mark Klein, *Evaluating Software Architectures: Methods and Case Studies*, Addison Wesley, 2002.

[47] *"ATAM: Method for architecture evaluation"*: ATAM - Architecture Trade-off Analysis Method report: http://www.sei.cmu.edu/ata/ata_method.html

[48] Rick Kazman, Len Bass, Gregory Abowd, and Mike Webb, "SAAM: A Method for Analyzing the Properties Software Architectures," Proceedings of the 16th International Conference on Software Engineering, Sorrento, Italy, May 1994, pp. 81-90. http://www.sei.cmu.edu/ata/publications.html#reports

[49] *"CBAM: Cost Benefit Analysis Method* http://www.sei.cmu.edu/ata/products_services/cbam.html

[50] Thomas J. Dolan, Ph.D. Thesis, *"Architecture Assessment of Information-System Families"*, Department of Technology Management, Eindhoven University of Technology, February 2002.

[51] P.Shanmugapriya , R.M.Suresh ,A Knowledge Based Approach to Enhance Software Architecture Review Process, *International Journal of Information Technology and Knowledge Management July-December 2012, Volume 5, No. 2, pp. 315-318*