A Framework for Classifying and Comparing Software Architecture Evaluation Methods

Muhammad Ali Babar, Liming Zhu, Ross Jeffery
National ICT Australia Ltd. and University of New South Wales, Australia

Abstract

Software architecture evaluation has been proposed as a means to achieve quality attributes such as maintainability and reliability in a system. The objective of the evaluation is to assess whether or not the architecture will lead to the desired quality attributes. Recently, there have been a number of evaluation methods proposed. There is, however, little consensus on the technical and non-technical issues that a method should comprehensively address and which of the existing methods is most suitable for a particular issue. This paper presents a set of commonly known but informally described features of an evaluation method and organizes them within a framework that should offer guidance on the choice of the most appropriate method for an evaluation exercise. In this paper, we use this framework to characterise eight SA evaluation methods.

1. Introduction

Software Architecture (SA) has been attracting tremendous attention from researchers and practitioners since the last decade of 20th century. The increasing size, complexity, and demand for quality software systems are some of the most important issues that have increased interest in this sub-discipline of software engineering. The size of software systems has increased ten fold over the past decade or so. The problems of developing and maintaining large systems have led to a realization that high level design description can play an important role in successfully understanding and managing large and complex software systems [1, 2]. During recent years, researchers and practitioners have also recognized that quality attributes (such as maintainability, reliability, usability, performance, flexibility etc.) of large software systems are largely constrained by the systems' SA [3].

Since SA plays a significant role in achieving system wide quality attributes, it is very important to evaluate a system's architecture with regard to desired quality requirements as early as possible. The principle objective of SA evaluation is to assess the potential of the chosen architecture to deliver a system capable of fulfilling required quality requirements and to identify any potential risks [4]. Additionally, it is quicker and less expensive to detect and fix design errors during the initial stages of the software development. That is why an effective SA assessment method to analyze prospective architectures is of great business value [5].

A number of methods have been developed to evaluate quality related issues at the SA level. The SA evaluation methods specifically studied in this paper are: Scenario-based Architecture Analysis Method (SAAM) [6], Architecture Tradeoff Analysis Method (ATAM) [7], Active Reviews for Intermediate Design (ARID) [8], SAAM for Evolution and Reusability (SAAMER)[1] [9], Architecture-Level Modifiability Analysis (ALMA) [10] , Architecture-Level Prediction of Software Maintenance (ALPSM)[1] [11], Scenario-Based Architecture Reengineering (SBAR)[1] [12], SAAM for Complex Scenarios (SAAMCS)[1] [13], and integrating SAAM in domain-Centric and Reuse-based development (ISAAMCR)[1] [14]. These are scenario-based methods, a category of evaluation methods considered quite mature. There are also some attribute model-based methods and quantitative models for SA evaluation (for example, [15-17] etc.), but, these methods are still being validated and are considered complementary techniques to scenario-based methods.

There is, however, little consensus on the technical and non-technical issues that a method should fully address and which of the existing methods is most suitable for a particular issue. There is not much work done on systematic classification and comparison of the existing methods. Moreover, there is not much guidance on the desirable features of the evaluation methods and their usefulness. Two research groups have attempted to address informally some of the above mentioned issues by providing a conceptual framework for method comparison [1, 18]. Each of these efforts enhances our understanding of the available methods; however, for reasons mentioned in section 2, each has significant limitations. Therefore,

[1] The abbreviated names are not what methods are originally called. We are using them for convenience.

we decided to survey the state of research to identify a set of commonly known but informally described features of an evaluation method and organize them within a framework that should offer guidance on the choice of the most appropriate method for an evaluation exercise. The main objective of this paper is to propose a framework to compare and evaluate various SA evaluation methods and demonstrate the use of the framework to discover the similarities and differences among existing methods.

The rest of the paper is structured as follows: section 2 discusses the contributions and limitations of previous surveys. Section 3 and its subsections present and discuss our framework along with rationale for selecting its components and compares the eight SA evaluation methods. Section 4 presents conclusions and future work.

2. Background work

Any attempt to provide a taxonomic comparison based on a comprehensive overview of the state-or-the-art in a particular area of research and practice is normally based on discoveries and conclusions of other researchers and practitioners and other previous surveys. We have made every effort to find and examine all the survey work done on SA evaluation methods during the last decade. To the best of our knowledge there is only one attempt [18] that provides a comprehensive treatment of the topic. None of the other published surveys or comparison of SA evaluation methods provides an explicit framework for comparing the methods. Instead quite often, these surveys have been published to support the need for developing a new evaluation method. For example, Bahsoon and Emmerich included an overview of the available SA assessment methods in their seminal work on ArchOptions [19]. Pual Clements et al. wrote a chapter on methods comparison in [1], however, in this comparison they only included three evaluation methods (SAAM, ATAM, and ARID [8]), all developed by the Software Engineering Institute. Furthermore, their comparison framework does not include some important features that an evaluation method should have, e.g., architecture definition, architectural description, tool support, and so forth.

We regard [18] as a first comprehensive attempt to provide a taxonomy of this growing area of research and practice; and it is quite valuable in better comprehending the existing SA evaluation methods. However, this work has its own shortcomings. For example, the authors do not provide any detailed explanation for the components of their comparison framework, nor do they explicitly describe the reasons for including those particular components in their framework. We assume that the elements of their framework are the features that they consider should be included in a method. We argue that an evaluation method should provide many more features (support for this argument is provided in section 3). Moreover, there have been significant advances in SA evaluation research since their work was completed three years ago. For example, assessment methods for non-traditional quality attributes (usability, stability etc.) are being developed. Other evaluation methods (ATAM and ARID) have been published in books [1, 20] along with a number of case studies.

3. Framework for classifying and comparing SA evaluation methods

As noted in the previous section, there has been little research on explicitly classifying and comparing SA evaluation methods. Therefore, we decided to develop a classification and comparison framework by discovering commonalities and differences found among the existing scenario-based SA evaluation methods. To a great extent, our framework includes features either supported by most of the existing methods or reported as desirable by SA researchers and practitioners.

To identify the components of our framework, we have also drawn upon a number of other sources including previously developed comparison frameworks for evaluation methods [1, 18], an extensive survey of SA literature, and an analysis of heuristics of experienced software architects and software engineers. We do not claim that we have produced an exhaustive list of features that a comparison classification and comparison framework for SA evaluation methods should have. This framework is quite easily modifiable as is necessary in a research area that is still in its inception stage [21]. Before describing and discussing each element of our classification and comparison framework, we provide definition of the most important concepts: software architecture, quality attributes and scenarios.

There is no standard, unanimously accepted definition of SA, we will use the definition provided by Bass et al. [20]: *"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."* This definition is mainly concerned with structural aspect of a system.

A quality attribute is a non-functional requirement of a software system, e.g., reliability, modifiability, performance, usability and so forth. According to [22], software quality is the degree to which the software possesses a desired combination of attributes. There are a number of classifications of quality attributes. In [23], McCall listed a number of classifications of quality attributes developed by a number of software engineering researchers including he himself. A later classification of software quality is provided in [24].

There are a number of definitions of scenarios as well. We will use as our working definition of scenario provided in [25]: "*A scenario is a brief description of a single interaction of a stakeholder with a system.*"

The following subsections motivate and elaborate on each of the components (given in right hand-side of Table 1) by providing a brief description along with the rationale for including it in the framework. Whilst Table 2 provides a comparison of the eight methods.

3.1 Maturity stage

SA evaluation is relatively a new area of research, which has gathered momentum during the last decade of 20th century. Redwine and Riddle [26] found that it usually takes 15-20 years for a technology to start getting popular support after going through 6 phases of development. Instead of using their model of technology maturation, we have conjectured that existing evaluation methods may be in any of the following four maturity phases: 1) inception (method recently surfaced and has not been validated yet for example [27]), 2) development (methods is continuously being validated with results appearing in credible literature), 3) refinement (method has been validated in various domains and refinements are continuously being made to various techniques, e.g. ATAM, ALMA etc.), and 4) dormant (method has one or few publication or no validation performed, or merged with other methods, no follow up etc. [17]). We do not have any formal model to substantiate this conjecture; it is based on a number of factors such as validation results, recent publications, number of case studies, reported in books, applicability to different classes of software systems, continuously being followed up and so forth. We believe that a method's maturity stage is important to foster confidence in method users.

Only two of the surveyed methods, ATAM and ALMA, can be considered in the refinement stage. These two methods have been used and validated in various domains. Refinements have been made to their techniques and process. Case studies using ATAM have been recently published in two books [1, 20]. SAAM and ARID can be considered in development stage for entirely different reasons. The former is being replaced by its descendant, ATAM, the later is yet to be fully validated. The rest of the surveyed methods are dormant or have been merged with other methods.

3.2 Definition of software architecture

A precise and well-documented definition of SA is very important for any SA evaluation activity to be performed successfully [28]. It is difficult to define metrics to assess the capability of SA with respect to quality attributes without precisely describing what SA means according to a particular evaluation method or framework [29]. Unfortunate, there are only a few architectural evaluation approaches (e.g. [28]) that provide a precise and clear definition of SA. Most of the well-known methods like ALMA and ATAM leave SA undefined under the assumption that every one knows what it means. Since the developers of these methods use the definition of SA provided in [20], the assumption might have been that all the users of the methods will also adopt the same definition as the working definition. However, this is not a very prudent approach when it is generally acknowledged fact that there is no standard definition of SA [30].

3.3 Process support

SA evaluation requires a number of activities to be performed. It needs a number of inputs, generates important artefacts, and requires stakeholders' participation. Like any process, the successful management of an SA evaluation requires careful planning, implementation, measurement, and control [31]. The role of this process is widening as SA researchers and practitioners are realising that architectural analysis is inseparable from architectural design. They claim that a design should be validated as early as possible and analysis techniques at the architectural level can provide important insight [20]. Some of the SA evaluation methods are being promoted as architectural design and analysis methods, e.g., ATAM [20] and Quality Attribute-oriented SA design method (QASAR) [32].

Like any other good software development method, a SA evaluation method should provide a set of standards, guidelines and heuristics for what needs to be done to achieve the objectives of the method. The method should provide explicit guidelines on when certain actions are to be taken and in which sequence, what are the inputs and deliverable. A method is significantly enhanced by good process support. Process support requires process definition and a framework to monitor and control the process [33].

Most methods provide at least coarse-grained description of the evaluation process. However, detailed guidance on various actions and techniques is sparse. Only ATAM and ARID provide sufficient process instructions. While SAAMER provides a framework to successfully steer the process. The remaining methods have process support embedded within the method description.

3.4 Method's activities

As we mentioned in the discussion on process support, there are normally a number of activities involved in a SA

evaluation exercise. The number, nature, and sequence of the activities usually vary from method to method. Additionally, the activities of the methods also differ in level of complexity and granularity. In scenario-based methods, there are a number of activities that appear to be same at the coarse-grained level; however, a fine-grained analysis of those activities reveals a number of differences. For example, scenario development and scenario evaluation activities are common in scenario-based methods, but the techniques of performing these activities are quite different. Moreover, some activities may be aggregation of other activities. For example, ATAM has four phases and there are a number of activities called steps in each of the phase. SAAMER provides a framework of four activities; one of its four activities has 6 sub-activities. Since SA evaluation methods can be compared based on fine-grained details of their respective activities, it is vital to provide detailed information on all aspects of the specific activities of a particular method.

Most of the activities of the studied methods vary in number, complexity and granularity. We can identify at least two common activities in all of the scenario-based methods, i.e., scenario generation and scenario evaluation. Most of the methods differ based on the techniques of arriving at the scenarios and performing the impact analysis. For example, SAAMCS provides an instrument to measure the complexity; ALMA uses scenario profiles to categorised the generated scenarios; ATAM provides a six element framework to characterise quality attributes, uses a utility tree for generating and classifying scenarios based on various quality attributes for evaluation.

3.5 Method's goals

SA evaluation can be performed for a number of purposes, e.g., risk assessment, maintenance cost prediction, architecture comparison, trade-off analysis and so forth. No one method can be considered as equally good for all types of assessment objectives as different methods are optimised to achieve different evaluation goals. The common goal of most of the evaluation methods is to evaluate the potential of the designed architecture to facilitate or inhibit the achievement of the required quality attributes. For example, some architectural styles, e.g., layered architectures, are less suitable for performance sensitive systems, even though they usually result in highly flexible and maintainable systems [32]. In order to achieve maximum benefit from an assessment activity, the goals of the evaluation need to be explicitly defined [34]. The goals of assessment help software architect make a number of critical decisions with regard to selection of a specific method and deliverable required.

There is at least one common goal found in all surveyed methods, which is prediction-based assessment of the quality of a system at the architecture level. However, each method has a specific view and different approach to achieve the goal: SAAM and its variants (specifically SAAMCS and ISAAMCR) are mainly geared to identify the potential architectural risks. However, SAAMCS focuses on exposing the boundaries of SA with respect to flexibility using complex scenarios, while, ISAAMCR integrates SAAM in domain-centric reusable development process; SAAMER evaluates the designed SA for evolution and reusability and provide a framework for SA analysis; ALMA specializes in predicting one quality attribute (i.e., modifiability) and there are three possible objectives to be pursued: risk assessment, maintenance cost prediction, and SA comparison; ARID performs suitability analysis of intermediate design artefacts; ATAM identifies and analyses sensitivity and trade-off points as these can prevent the achievement of a desired quality attribute.

3.6 Number of quality attributes

Different software engineering communities have developed different techniques for characterising their respective quality attributes and the methods to evaluate software systems with respect to that particular quality attribute, e.g., real-time [35], reliability [36], and performance [37]. These assessment techniques study a specific quality attribute in isolation. In reality, however, quality attributes interact with each other. For example, there is generally a conflict between configurability and performance [38]; performance also impacts modifiability, availability affects safety, security conflicts with usability, and each quality attribute impacts cost [7]. That is why it is important to find an appropriate balance of quality attributes in order to develop a successful product.

One of the most significant features of method differentiation and classification is the number of quality attributes a method deals with. Most of the surveyed methods focus mainly on a single quality attribute. For example, ALMA is aimed at architectural-level modifiability analysis, while SAAM was also initially developed to assess modifiability but later on it became a generic SA evaluation method for any quality attribute to be considered in isolation. However, SA analysis of a large complex system usually requires multi-attribute analysis. Currently, SBAR and ATAM are the only methods that consider multiple quality attributes. SBAR also classifies the quality attributes into development oriented, e.g., reusability and maintainability, and operational related, e.g., reliability and performance [12]. ATAM specifically focuses on architectural decisions that affect (positively or negatively) one or more quality

attributes, which are called either sensitivity or trade-off points depending upon the number of attributes affected by an architectural decision.

3.7 Applicable project stage

SA evaluation is traditionally performed after the specification of the architecture and before the beginning of the implementation. There are situations in which post-deployment architectural assessment needs to be performed. For example, the architecture of a legacy system may be evaluated for reengineering. Abowd et al. [5] described three phases of interest to architectural evaluation: early, middle, and post-deployment. Maranzano [39] reported two forms of architectural review at AT & T: *architectural discovery* (quite early) and *architecture review* (before coding).Thus, it is obvious that different organizations may hold architectural evaluation at different stages of the development lifecycle, i.e., some quite early, others quite late. The rule of thumb is that an architectural assessment should be held as soon as architecture related decisions are beginning to be made and the cost of reversing those decisions would be more than conducting the evaluation [1].

SA evaluation methods can be classified by considering the development stage of their applicability. From this perspective, most of the surveyed methods (SAAM, SAAMCS, ISAAMCR, and ALMA) are applied to the final version of the SA. Although, ATAM is usually applied to the final version of SA, it is also used as an architecture analysis and design improvement method in architecture-based development process. SBAR is another architecture analysis and design transformation method usually applied iteratively. ARID is used to analyse the suitability of various components when SA architecture is still in its inception stage.

3.8 Architectural description

Communicating SA to its stakeholders is one of the critical factors of a successful SA evaluation exercise. That is why it is important to use an appropriate notation and abstraction level to capture the architecture [40]. SA researchers and practitioners have developed a number of Architectural Description Languages (ADLs) with little consensus on a universally accepted definition of ADL. A common understanding is that an ADL is a formal language used to represent the architecture of a software-intensive system [21]. Since architectural description has a vital role in the architecture assessment process, an evaluation method should guide its user on which ADL is most appropriate to be used.

The role of architectural views is also considered vital in communicating architectural information. An architectural view is considered to be a perspective that satisfies the expressed needs of a stakeholder. The importance of multiple views as an effective means of separating the concerns during architectural design and analysis has been realised by SA researchers and practitioners in [20, 41, 42]. However, there are different opinions on the number and nature of the architectural views (for example Hofmeiser et al. [41] suggest conceptual, module, execution, and code views; Kruchten [42] gave 4+1 set of views: logical, process, implementation, deployment and use case views; Len Bass et. al. [20] provided a more extend list of architectural views). When there is no “canonical” set of views, it is very important for the evaluation method to provide explicit guidelines as to which views convey the most important information required.

Table 1. SA evaluation method classification and comparison framework

Framework components	Brief explanation
Maturity stage	What is the level of maturity (inception, development, refinement or dormant)?
Definition of SA	Does the method explicitly consider a particular definition of SA?
Process support	How much support is provided by the method to perform various activities?
Method’s activities	What are the silent activities to be performed and their order to achieve the objectives of the method? Which activities are common in various methods?
Method’s goals	What are the particular goals of the methods?
Number of quality attributes	How many and which quality attributes are covered by the method?
Applicable project stage	Which is the most appropriate development phase to apply the method?
Architectural description	What form of architectural description is recommended (e.g., formal, informal, particular ADL etc.)?
Evaluation approaches	What types of evaluation approaches are included in the method?
Stakeholders involvement	Which groups of stakeholders are required to participate in the evaluation?
Support for non-technical issues	Does method provide any support/guidance on non-technical (such as social, organisational, managerial, or business) issues of evaluation process?
Method’s validation	Has the method been validated? How has it been validated?
Tool support	Does method provides or recommend any tool for all or some of the tasks?
Experience repository	Does method recommend any experience repository? What is the level of producing and use reusable knowledge?
Resources Required	How many man-days are required? What is the size of evaluation team?

All of the surveyed methods recognise the importance of an appropriate architectural description for successful evaluation; but they are independent of any architectural description language (ADL). A method's users can use any ADL that is comprehendible by all participants. Different SA methods require the SA architecture be described at various levels of abstraction, e.g., micro, macro, high-level, detailed etc. For example, SAAMCS divides the SA into micro and macro architecture for change impact analysis. ARID needs detailed design of the components. Most of the studied methods require several architectural views for the evaluation exercise. Some of them need one or two, others require a wide variety, depending on the groups of stakeholders involved. For example, logical and module views may suffice for SAAM, but process, data-flow, user, physical, module and many more may be required for ATAM. ALMA and SAAMER also require several views, e.g., contextual, conceptual, dynamic, etc.

3.9 Evaluation approaches

SA researchers and practitioners have been using a wide range of approaches to analyse candidate architectures with respect to desired quality attributes. Abowd et al. [5] and Bass et. al. [20] have classified the analytical techniques into two main categories: questioning and measurement. The former category includes analysis techniques like scenarios, questionnaire, and check lists; these techniques aim to generate qualitative questions concerning an architecture's ability to support a particular quality attribute or combination of quality attributes [43]. The later category includes metrics, simulations, prototypes, and experiments. Abowd et al. [5] also provided a four dimensional framework to further categorise them. The four dimensions are generality, level of detail, phase, and what is evaluated.

Bosch [32] and PerOlof [10] have also informally classified evaluation approaches. Their categories of assessment techniques include scenario-based, simulation or prototyping, mathematical modelling, and experience-based. Bosch and Molin [44] have classified quality attributes into two main categories: development (e.g. maintainability, flexibility) and operational (e.g. performance, reliability). Scenario-based and experienced-based techniques are considered more appropriate for development quality attributes; and simulation or prototyping and mathematical modelling are recommended for operational quality attributes [11]. Although, most of evaluation methods heavily rely on techniques classified in one of the above-mentioned categories, the need to use a combination of both qualitative and quantitative techniques is recognized [1, 32]. In order to select an appropriate evaluation method, it is important to know which techniques are included, what level of information required and which stage is the most appropriate to apply the method. We believe a good method should explicitly answer these questions.

SA evaluation methods can be classified based on their analysis techniques: some are pure scenario-based (SAAM, SAAMCS), a few use a variety of approaches depending on the attribute being studied or evaluation goal (SBAR, ALMA), others combine scenarios with attribute model-based analysis (ATAM) or with active design reviews (ARID). Moreover, some methods provide analytical models for quality attributes (ALMA); others use those provided by various quality attribute communities. ATAM is quite flexible method for integrating existing theoretical models of attributes.

3.10 Stakeholder involvement

A stakeholder is any person or organisational representative who has a vested interest in the results of a software development project [45]. The system built on a particular architecture serves a wide range of interests of a multitude of stakeholders, i.e., end users, managers, developers, customers, architects, testers or anyone else whose cares about the outcomes of the project. Since these stakeholders have a stake in a system's architecture, all of them may attempt to influence the architectural design decision in various ways. For a successful architectural evaluation, it is important to identify all key stakeholders along with their objectives. A stakeholder-centric analysis approach considers each stakeholder as a context [28].

Clements et al. [1] describe the active participation of stakeholders in the evaluation process as absolutely essential for a high-quality evaluation. Parnas [34] regards the presence of wrong people in the design review sessions as the one of the major problems with the conventional design review approaches. This is also true of architectural evaluation sessions. The goals of the evaluation must be made clear in order to get stakeholders to "buy-in" to the process [1]. A method should describe the required stakeholders along with the selection criteria to be used to identify them.

The role of stakeholders is vital in scenario-based methods because scenarios are mainly generated by the stakeholders. There are also stakeholder-centric analysis approaches like [28]. All the studied methods recognise the importance of involving appropriate stakeholders. However, the number and types of stakeholders involved vary from method to method. For example, SAAM and its variants like SAAMCS and SAAMER involve all major stakeholders, e.g., architects, designers, end users, customers, developers etc., while SBAR requires the architecture designer to evaluate an implemented architecture for reengineering without involving any other stakeholders, which obviously needs less time and fewer

resources. ALMA involves various stakeholders for different activities. ATAM requires the involvement of a multitude of stakeholders and the evaluation team.

3.11 Support for non-technical issues

Software systems are not isolated entities; rather, they are used in social and organisational contexts. The process used to develop these software systems is a social process [46]. Since SA design and evaluation is a significant part of the software development process, it should also be considered as a social process. SA design and evaluation are greatly influenced by non-technical issues like organisational structure, organisational communication channels, stakeholders' vested interests, psychological and political factors, managerial concerns etc. Thus, sociological issues are important because most of the well-established evaluation methods are scenarios-based; and scenarios usually come from the stakeholders. Moreover, being external to the project, an evaluation team is not usually fully trusted [1]. However, these non-technical issues have not been given appropriate attentions until relatively recently. The importance of appropriately addressing non-technical issues is becoming more apparent in recent validation efforts [40, 47].

Only a handful of the existing methods explicitly consider and sufficiently address social aspects of the evaluation exercise. ATAM stands out from its counterparts in terms of its detailed guidelines and techniques to deal with social issues. It provides a common framework for the interaction among the stakeholders and domain experts, making implicit assumptions explicit during analysis, and negotiating architectural tradeoffs objectively. Some methods briefly mention social issues without suitably dealing with them (SAAM, SAAMER); others completely ignore these issues.

3.12 Method's validation

Developing an evaluation method based on personal experiences, general observations, heuristics of the practitioner, or published best practices and claiming that it will work is not good enough. A method developer should validate the method by applying various means of experimental validation available in software engineering domain. The process of method development and the techniques used to validate it may encourage or discourage the evaluators to select one particular method over the other [48]. Thus, it is necessary to validate the SA evaluation methods by applying them to a number of large and complex software-intensive applications in various domains to demonstrate that their general applicability.

Most of the methods, discussed in this paper, have been validated by applying them to different research and commercial applications. However, the number of applications and variety of domains used to validate them vary considerably. For example, the applicability of the methods like SAAM, ATAM, and ALMA have demonstrated by using them for evaluating the SA of large systems like battlefield control system and the Earth Observation System. These methods are being refined as a result of feedback from their use.

3.13 Tool support

SA design and evaluation is a human and knowledge intensive process, which has a number of tedious and expensive tasks like collecting, documenting, and managing the information relevant to architectural design and evaluation decisions. The overheads of such tasks are considered one of the impediments to the wide spread adoption of architectural design and evaluation approaches in the industry [49]. That is why the SA community has been emphasising the need to automate as many of the tedious, expensive, and error-prone tasks of SA design and evaluation as possible [25]. A tool can also capture the design artefacts along with the decision rationale, evaluation outcomes, measurement and administrative information that are invaluable assets for future.

Despite the recognition of the importance of tool support by almost all the surveyed methods, only SAAM provides a tool (SAAMTOOL) to partially support the evaluation process. This tool was briefly introduced in [49] along with a discussion of the desired features that a SA design and analysis tool should provide. However, to the best of our knowledge, there is no other published material on SAAMTOOL.

3.14 Experience repository for reusable knowledge

Knowledge management for reusability is a relatively new area of research and practice. However, the notion of reuse has long been recognized as one of the most important means of gaining productivity, quality and cost-effectiveness in the software engineering domain [50]. In software development, the idea of reusability is generally associated with code reuse. However, reusability is a much wider notion that goes beyond APIs and class libraries to include many types of artefacts of the software development project, e.g., requirements specification, design patterns, architectural description, design rationale, document templates, test scripts and so forth [51]. Like any other software development activity, architectural evaluation is a human and knowledge-intensive activity that can be a very expensive practice if each evaluation starts from scratch. Instead of reusing knowledge from the previous activities in an ad-hoc way, activities necessary to manage and utilize the artefacts produced by the previous

SA evaluation efforts must be explicitly incorporated in SA method.

Only ATAM and ESAAMI explicitly incorporate reusable knowledge in the SA evaluation process. ESAAMI suggests that analysis templates be developed that include different reusable assets like proto-scenarios, classification hints, proto-evaluation, and weights [14]. However, there is no explicit guidance on how to store, manage, and retrieve these reusable assets, nor does it specifically emphasize the need for an experience repository. ATAM provides a comprehensive guidance on generating and utilizing the reusable artefacts, i.e., identified risks, scenarios, quality attributes, attribute-based architecture styles etc. It also recommends maintaining repositories of these artefacts and updating them regularly [1].

Table 2. The comparison of various evaluation methods using the framework

Methods / Components	SAAM	ATAM	ARID	SAAMER	ALMA	SBAR	SAAMCS	ISAAMCR
Maturity stage	Refinement/ Dormant	Refinement	Development	Dormant	Refinement/ Development	Dormant or merged	Dormant or merged	Dormant or merged
SA Definition	Left to users	Left to users	Left to users	Left to users	Not provided	Not provided	Not provided	Not provided
Process support	Not explicitly addressed	Comprehensively covered	Sufficient process support	Framework to support evaluation	Increasingly being provided	Embedded in method description	Steps explained but insufficient guidance	Coarse-grained description
Method's activities	6 activities	9 activities in 2 phases	9 activities in 2 phases	4 activities carried out iteratively	5 activities carried out sequentially	3 activities carried out iteratively	3 activities, 2 need parallel execution	6 activities, 4 need reusable artefacts
Method's goals	Risk identification Suitability analysis	Sensitivity & Trade-off analysis	Validating design's viability for insights	Assessing SA for reuse and evolution	Change impact analysis, predicting maintenance effort	Evaluate ability of SA to achieve quality attributes	Predicting context relative flexibility, risk assessment	Analysing flexibility for reusability
Quality attributes	Mainly Modifiability	Multiple attributes	Suitability of the designs	Reusability/ Evolution	Maintainability	Multiple attributes	Flexibility/ Maintainability	Flexibility reusability
Applicable project stage	Once functions assigned to modules	After SA/detailed design, iterative process	During component design	Implemented architecture	During design phase	system extension or reengineering stage	Before implementing architecture design	Like SAAM on final version of SA
Architectural Description	Logical & module views required	Process, data-flow, uses, physical & module views	Preliminary design in sufficient details with appropriate views	Static, Map, Dynamic, Resource views required	Independent of architectural description and notation used	An implemented architecture	Micro & Macro architecture designs	Architectural views analysis templates
Evaluation approaches	Scenario-based functionality and change analysis	Hybrid approach: questioning & measuring	Combination of scenario-based & active design reviews	Information modelling & scenarios-based	Depending on the analysis goal	Multiple approaches	Scenario-based (complex scenarios)	Scenario-based
Stakeholders involved	All major stakeholders	All major stakeholders	Designer, architect, and anyone interested	Designer, manager and end-users	Various for different activities	Architecture Designer	Involvement of major stakeholders implied	All major stakeholders
Support for non-technical issues	Such issues briefly mentioned	Sufficiently provided	Provided as ATAM & ADR support these issues	A little consideration paid	Not explicitly addressed	Not considered	Implicit recognition of ownership issues	Briefly mentioned like in SAAM
Method's validation	Validated in various domains	Continuously being validated	Not extensively validated yet	Applied on one switching system	Validated in different domains	Validated on one software system	Not results reported on validation	No published record of validation
Tool support	Partially available	Not available	Not available	Not available	Not Available	Not available	Not available	Not available
Experience repository for reusable knowledge	Not available	Repository mainte-nance is highly recommended	Not considered	Not available	Not available, Historical data is required.	Not considered	Not considered	Analysis templates, reusable artefacts.
Resources Required	Apart from initial & post preparation, 3 days. 4-person evaluation team & stakeholders	Apart from initial & post preparation, 2 days & 3-person evaluation team & stakeholders	Apart from initial & post preparation, 2 days & 2-person evaluation team & stakeholders	Not specified	Not specified	Not specified	Not specified	Not specified

3.15 Resources required

SA evaluation needs a number of organisational resources and normally incurs significant cost; that is why it is normally recommended for large software intensive systems. Evaluating SAs for large-scale systems is a complicated and challenging task that may require substantial organisational resources [20]. For example, based on 300 full-scale architecture evaluation on projects requiring a minimum of 700 staff days, AT & T reported an average cost of 70 staff days, while Rational Software Corporation charged an average cost of $5000 dollars for projects of at least 500 thousands lines of code [5]. However, the resources required and cost incurred tends to decrease over time as an organisation develops a culture of evaluation and reuse at the architectural level.

Most of the surveyed methods do not provide any information on the cost of an evaluation or the resources required. Some methods (e.g., SAAM, ATAM, and ARID) mention the desirable shape of the evaluation team and various stakeholders, however, there is hardly any information about other resources or the cost of using these methods.

4. Conclusions and future work

The main contribution of this paper is to present a framework for classifying and comparing SA evaluation methods. This framework has been developed by discovering similarities and differences among existing evaluation methods. We have also demonstrated how the proposed framework can be used to identify the essential features that a good evaluation method should provide.

The comparison of existing methods revealed several features supported by most of the well-established methods. For example, a number of methods provide suitable guidance on required architectural description and views. Most of them also provide appropriate techniques for quality attribute charactersition and scenario generation and evaluation. On the other hand, the survey also highlighted a number of issues which existing method do not sufficiently address. Only one method, ATAM, provides comprehensive process support. Social aspects of the evaluation have been given quite sparse attention. No working definition of the SA is explicitly provided. Finally, tool support for the evaluation process is almost non-existent. Thus, the proposed framework provides an important advance towards answering questions regarding the various features a good SA evaluation method should have and why and how to compare evaluation methods.

We do not suggest that our framework is complete in its existing form. Rather, we expect this framework to be modified and extended in the future as a result of our ongoing research. Presently, we are contemplating several issues. In particular embedding SA evaluation activity in a development process and developing various techniques and tools to support and improve the evaluation process. We believe these issues indicate some of the areas where future research in this domain should be concentrated.

5. Acknowledgements

The authors like to thank Dr. Aurum and Prof. Kitchenham for their comments on draft of this paper.

6. References

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

[2] C.-H. Lung and K. Kalaichelvan, "An Approach to Quantitative Software Architecture Sensitivity Analysis," *International Journal of Software Engineering and Knowledge Engineering*, vol. 10, No. 1, 2000, pp. 97-114.

[3] P. Bengtsson, "Towards Maintainability Metrics on Software Architecture: An Adaptation of Object-Oriented Metrics," First Nordic Workshop on Software Architecture, Ronneby, 1998.

[4] N. Lassing, D. Rijsenbrij, and H. v. Vliet, "The goal of software architecture analysis: Confidence building or risk assessment," Proceedings of First BeNeLux conference on software architecture, 1999.

[5] G. Abowd, L. Bass, P. Clements, R. Kazman, L. Northrop, and A. Zaremski, "Recommanded Best Industrial Practice for Software Architecture Evaluation," SEI, Carnegie Mellon University CMU/SEI-96-TR-025, 1997.

[6] R. Kazman, L. Bass, G. Abowd, and M. Webb, "SAAM: A Method for Analyzing the Properties of Software Architectures," Proceedings of the 16th International Conference on Software Engineering, 1994.

[7] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, and J. Carriere, "The Architecture Tradeoff Analysis Method," Proceedings of IEEE, ICECCS, 1998.

[8] P. C. Clements, "Active Reviews for Intermediate Designs," SEI, Carnegie Mellon University CMU/SEI-2000-TN-009, 2000.

[9] C.-H. Lung, S. Bot, k. Kalaichelvan, and R. Kazman, "An Approach to Software Architecture Analysis for Evolution and Reusability," Proceedings of CASCON, 1997.

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

[11] P. Bengtsson and J. Bosch, "Architectural Level Prediction of Software Maintenance," Proceedings of 3rd European Conference on Software Engineering Maintenance and Reengineering, 1999.

[12] P.-O. Bengtsson and J. Bosch, "Scenario-based Architecture Reengineering," 5th International Conference on Software Reuse, 1998.

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

[14] G. Molter, "Integrating SAAM in Domain-Centric and Reuse-based Development Processes," Proceedings of the 2nd Nordic Workshop on Software Architecture, Ronneby, 1999.
[15] M. Svahnberg, C. Wohlin, L. Lundberg, and M. Mattsson, "A Method for Understanding Quality Attributes in Software Architecture Structures," Proceedings of the 14th International Conference on Soft Engineering and Knowledge Engineering, Ischia, Italy, Jul., 2002.
[16] M. Klein and R. Kazman, "Attribute-Based Architectural Styles," Soft Engineering Institute, Carnegie Mellon University, Technical Report CMU/SEI-99-TR-022, October 1999.
[17] J. C. Duenas, W. L. d. Oliveira, and J. A. d. l. Puente, "A Software Architecture Evaluation Model," 2nd International Workshop On Development and Evolution of Software Architectures for Product Families, Feb. 1998.
[18] L. Dobrica and E. Niemela, "A Survey on Software Architecture Analysis Methods," *IEEE Transactions on Software Engineering*, vol. 28, No. 7, 2002, pp. 638-653.
[19] R. Bahsoon and W. Emmerich, "Evaluating Software Architectures: Development, Stability, and Evolution," Proceedings of ACS/IEEE Int. Conf. on Computer Systems and Applications, Tunis, Tunisia, July, 2003.
[20] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 2 ed, Addison-Wesley, 2003.
[21] N. Medvidovic and R. N. Taylor, "A Classification and Comparison Framework for Software Architecture Description Languages," *IEEE Transactions on Software Engineering*, vol. 26, No. 1, Jan, 2000, pp. 70-93.
[22] *IEEE Standard 1061-1992, Standard for Software Quality Metrics Methodology*, Institute of Electrical and Electronic Engineers, New York, 1992.
[23] J. A. McCall, "Quality Factors," in *Encyclopedia of Software Engineering*, vol. 2, J. J. Marciniak, Ed. New York, U.S.A.: John Wiley, 1994, pp. 958-971.
[24] ISO/IEC, *Information technology - Software product quality: Quality model*, ISO/IEC FDIS 9126-1:2000(E).
[25] R. Kazman, S. J. Carriere, and S. G. Woods, "Toward a Discipline of Scenario-based Architectural Engineering," *Annals of Software Engineering*, vol. 9, 2000.
[26] S. Redwine and W. Riddle, "Software Technology maturation," Proceedings of the 8th ICSE, 1985.
[27] E. Folmer, J. V. Gurp, and J. Bosch, "Scenario-Based Assessment of Software Architecture Usability," Proceedings of Workshop on Bridging the Gaps Between Software Engineering and Human-Computer Interaction, ICSE, Portland, 2003.
[28] S. Bot, C.-H. Lung, and M. Farrell, "A Stakeholder-Centric Software Architecture Analysis Approach," Proceedings of 2nd International Software Architecture Workshop, 1996.
[29] A. Avritzer and E. J. Weyuker, "Investigating Metrics for Architectural Assessment," Proceedings of the 5th International Software Metrics Symposium, Bethesda, Maryland, 1998.
[30] D. Garlan, R. Allen, and J. Ockerbloom, "Architectual Mismatch, or, Why it's hard to build systems out of existing parts," 17th International Conference on Software Engineering, Apr.1995.
[31] M. E. Fagan, "Design and Code Inspections to Reduce Errors in Program Development," *IBM Systems Journal*, vol. 15, No. 3, 1976, pp. 182-211.
[32] J. Bosch, *Design & Use of Software Architectures: Adopting and evolving a product-line approach*, Addison-Wesley, 2000.
[33] I. Graham, B. Henderson-Sellers, and H. Younessi, *The OPEN Process Specification*, Addison-Wesley, 1997.
[34] D. L. Parnas and D. M. Weiss, "Active Design Reviews: Principles and Practices," Proceedings of the 8th International Conference on Software Engineering, London, England, 1985.
[35] M. H. Klein, T. Ralya, B. Pollak, and R. Obenza, *A Practitioner's Handbook for Real-Time Analysis*, Kluwer Academic, 1993.
[36] M. R. Lyu, "Handbook of Software Reliability Engineering." New York: McGraw-Hill and IEEE Computer Society, 1996.
[37] C. U. Smith and L. G. Williams, "Software Performance Engineering: A Case Study Including Performance Comparison with Design Alternatives," *IEEE Transactions on Software Engineering*, vol. 19, No. 7, July, 1993, pp. 720-741.
[38] L. Lundberg, J. Bosch, D. Häggander, and P.-O. Bengtsson, "Quality Attributes in Software Architecture Design," Proceedings of the IASTED 3rd International Conference on Software Engineering and Applications, Oct., 1999.
[39] J. Maranzano, "Best Current Practices: Software Architecture Validation," AT&T, the USA 1991.
[40] M. Moore, R. Kazman, M. Klein, and J. Asundi, "Quantifying the Value of Architecture Design Decisions: lessons from the Field," Proceedings of the 25th International Conference on Software Engineering, 2003.
[41] C. Hofmeister, R. L. Nord, and D. Soni, *Applied Software Architecture*, Adison-Wesley Longman, Reading, MA, 2000.
[42] P. Kruchten, "The 4+1 View Model of Architecture," *IEEE Software*, vol. 12, No. 6, 1995, pp. 42-50.
[43] M. R. Barbacci, M. H. Klein, and C. B. Weinstock, "Principles for Evaluating the Quality Attributes of a Software Architecture," SEI, Carnegie Mellon University CMU/SEI-96-TR-036, 1996.
[44] J. Bosch and P. Molin, "Software Architecture Design: Evaluation and Transformation," IEEE Engineering of Computer-Based Systems Sympsium, 1999.
[45] P. Kruchten, *Rational Unified Process: An Introduction*, Addison-Wesley, 2000.
[46] I. Sommerville and T. Rodden, "Human, Social, and Organisational Influences on the Software Process," Cooperative Systems Engineering Group, Lancaster University 1995.
[47] R. Kazman and L. Bass, "Making Architecture Reviews Work in the Real World," *IEEE Software*, vol. 19, No. 1, 2002, pp. 62-73.
[48] M. Shaw, "The Coming-of-Age of Software Architecture Research," Proceedings of the 23rd International Conference on Software Engineering, 2001.
[49] R. Kazman, "Tool Support for Architecture Analysis and Design," Proceedings of the 2nd International Software Architecture Workshop, San Francisco, Carlifornia, Oct., 1996.
[50] A. Aurum, R. Jeffery, C. Wohlin, and M. Handzic, "Managing Software Engineering Knowledge." Berlin Heidelberg: Springer- Verlag, 2003.
[51] A. I. Wasserman, "Toward a Discipline of Software Engineering," *IEEE Software*, vol. 13, No. 6, Nov.1996, pp. 23-31.