UDC 614.2+574/578+004.38

COMPARISON OF SOFTWARE ARCHITECTURE EVALUATION METHODS APPLICABILITY IN THE CONTEXT OF CQRS WITH EVENT SOURCING ARCHITECTURAL VARIATIONS

Hruzin D. L. – Post-graduate student of the Department of Electronic Computing Machinery, Oles Honchar Dnipro National University, Dnipro, Ukraine. ROR: https://ror.org/00qk1f078. ORCID: 0009-0004-8534-2559.

Lytvynov O. A. – PhD, Associate Professor of the Department of Electronic Computing Machinery, Oles Honchar Dnipro National University, Dnipro, Ukraine. ROR: https://ror.org/00qk1f078. ORCID: 0000-0001-7660-1353.

ABSTRACT

Context. This study is conducted in the context of developing and justifying a methodology for software architecture (SA) evaluation in relation to the Command Query Responsibility Segregation (CQRS) with Event Sourcing (ES) architectural variations.

Objective. This work aims to evaluate and compare the applicability of SA evaluation methods to support the selection of an optimal CQRS with ES architectural variation for real-world projects.

Method. Various SA evaluation methods are applied to enhance objectivity in architectural decisions. However, these methods are not universal; they vary in depth, focus, and required effort. The task considered in this work is the selection among CQRS with ES architectural variations, often structurally similar and thus difficult to distinguish using general-purpose evaluation methods. Comparing architectural variations requires in-depth analysis; however, for most methods, practical implementation is limited by time and resource constraints. The proposed approach identifies the most appropriate SA evaluation method for selecting between CQRS with ES architectural variations. It is based on a validated framework for classifying and comparing SA evaluation methods. In addition to qualitative analysis, the approach introduces a quantitative assessment of applicability to a specific case, allowing for supporting more informed decision-making.

Results. The approach was applied to compare several SA evaluation methods, including Information Technology for Decision-making Support regarding CQRS with ES Architectural Variations (DSAV-CQRSES), a method specifically designed for evaluating variations of the CQRS with ES architecture.

Conclusions. The existing framework of comparing Software Architectures cannot be directly applied to architectural variations (the deviations of the architecture significant for customer). The proposed modifications of the framework are primarily focused on CQRS with ES variations assessment.

KEYWORDS: Software Architecture, Comparison of evaluation methods, CQRS with Event Sourcing, architectural variations.

ABBREVIATIONS

ADL is an Architecture Design Languages;

ALMA is an Architecture-Level Modifiability Analysis;

ATAM is an Architectural Trade-off Analysis Method;

CBAM is a Cost-Benefit Analysis Method;

CMMI is a Capability Maturity Model Integration;

CQRS is a Command Query Responsibility Segregation;

DSAV-CQRSES is an Information Technology for Decision-making Support regarding CQRS with ES Architectural Variations;

ES is an Evant Sourcing;

MCDA is a Multi-Criteria Decision Analysis;

NIMSAD is a Normative Information Model-based System Analysis and Design;

PASA is a Performance Assessment of Software Architecture;

RTP is a Representative Test Project;

SA is a Software Architecture;

SAAM is a Software Architecture Analysis Method;

SQUASH is a Systematic Quantitative Analysis of Scenarios' Heuristics.

NOMENCLATURE

AS_i is an alternative (architectural strategy) being evaluated;

Cont_{ij} is a contribution of AS_i to QA_j;

E is a vector that contains effectiveness values;

E_{asc} is a sorted vector that contains effectiveness values;

e_i is an effectiveness of the i-th method in vector E;

ea_j is an effectiveness of the j-th method in sorted vector E_{asc};

$E_{\max}$ is a sorted vector that contains the highest effectiveness values;

m is a number of criteria;

Max is a maximum possible distance from the etalon;

n is a number of SA evaluation methods considered in the comparison;

p is a vector representing the candidate method;

$QAscore_j$ is a weight assigned to QA_j;

q is a vector representing the reference method;

ref is a vector representing the reference method;

SA is a set of SA evaluation methods;

SA_{opt} is a set of optimal SA evaluation methods;

W_i is a weight of the i-th criterion;

σ is a sorting function by ascending value;

σ′ is an inverse function of σ;

DOI 10.15588/1607-3274-2026-1-10

χ is a transformation function from SA to E;

χ' is an inverse function of χ.

INTRODUCTION

Developing large-scale software applications is a complex and resource-intensive process. One of the key aspects of this process is the selection of the most appropriate software architecture (SA). In organizations with low maturity levels, architectural decisions are often made intuitively, primarily based on the prior experience of individual developers. At higher levels of organizational maturity, such as Level 4 of the Capability Maturity Model Integration (CMMI) model [1] (Quantitatively Managed Organization), software development companies are focused on the predictability of quantitative performance improvement objectives and well-justified choices regarding SA solutions.

Within the architectural solutions which represent high level of abstraction SA variations can be seen as deviations which arise from either structural modifications or the application of additional architectural solutions intended to address specific technical challenges while preserving the core principles of SA.

One of the architectures that has multiple variations is the Command Query Responsibility Segregation (CQRS) with Event Sourcing (ES) architecture. This architecture is typically used in software systems with complex structure and business logic. In such systems, even a small architectural change can lead to significant increase of required development effort. For instance, solving the causal event synchronization problem [2] can affect a number of already existing modules. To avoid unexpected expenses during the development of software systems based on CQRS with ES architectural variations, it is necessary to objectify the selection of not only the SA, but also its variation.

A number of methods have been proposed to evaluate and compare SA solutions [3]. One such approach is the Information Technology for Decision-making Support regarding CQRS with ES Architectural Variations (DSAV-CQRSES) method [4]. This raises the issue of identifying the most suitable method for comparison CQRS with ES architectural variations within the boundaries of development team and project requirements and limitations. Several studies [5–6] have attempted to address this issue by providing qualitative comparisons and classification frameworks for SA evaluation methods. Others [7–8] have analysed pairs of methods based on large-scale statistical surveys of their practical application. However, these comparisons are typically generic and do not account for the specificity of CQRS with ES architectural variations and the context of projects or development teams.

Based on practice experience of DBB Software [9] company, it is assumed that the DSAV-CQRSES method is the most appropriate candidate.

Since this study focuses on evaluating the applicability of the DSAV-CQRSES, which enables objective selection of a suitable CQRS with ES architectural variation, a number of scoping restrictions are defined:

– The analysis is limited to the CQRS with ES architecture and its variations.

– Methods for SA evaluation are categorized by application phase as design-time or run-time techniques, with some methods applicable in both phases [3]. This work considers methods that can be applied at the design stage.

– Also, according to [3], evaluation methods may be classified into utility-based, scenario-based, parametric-based, search-based, economics-based, and learning-based categories. This study focuses exclusively on scenario-based methods since the DSAV-CQRSES approach itself belongs to this category and is grounded in use case analysis.

This work provides a brief overview of several SA evaluation methods selected based on a systematic literature review analysis [3, 10–11]. It considers existing approaches for classifying and comparing these methods. It also introduces an SA variation-oriented approach, based on the framework for classifying and comparing SA evaluation methods, which enables the transformation of qualitative assessments into quantitative, thereby facilitating the identification of the most suitable SA variation evaluation method.

The object of study is the process of comparing and assessing the applicability of SA evaluation methods.

The subject of study is methods, approaches, and frameworks for comparing and selecting the most suitable SA evaluation method to assess CQRS with ES architectural variations at the system design stage, as well as to provide an effective strategy for their evolution.

The purpose of the work is to evaluate and compare the applicability of SA evaluation methods to support the selection of an optimal CQRS with ES architectural variation within the context of a specific software company and its real-world projects.

1 PROBLEM STATEMENT

A wide range of architectural choices exists, ranging from structural and organizational aspects, such as choosing between monolithic and microservices architectures, to conceptual paradigms including Event Sourcing, Domain-Driven Design, and Service-Oriented Architecture.

To determine the most suitable solution, various methods are proposed, including Architectural Trade-off Analysis Method (ATAM) [12], Software Architecture Analysis Method (SAAM) [13], and DSAV-CQRSES [4], among others. Given the substantial number of available approaches, the question of selecting the most appropriate evaluation method becomes relevant. Several studies [5, 7, 11] have attempted to compare such methods. However, these comparisons are typically qualitative in nature and not designed to support the evaluation of architectural variations.

© Hruzin D. L., Lytvynov O. A., 2026
DOI 10.15588/1607-3274-2026-1-10

Suppose given a set of scenario-based SA evaluation methods $SA=\{sa_1, sa_2, \ldots, sa_n\}$ consolidated from a comprehensive literature review and primary research sources, as well as project-specific requirements, priorities, and constraints provided by the decision-making team in the form of (i) a vector of attribute weights *w* and (ii) a vector of optimal (target) attribute values *ref*.

Thus, the problem – evaluating the applicability and effectiveness of scenario-based SA evaluation techniques for comparing CQRS with ES architectural variations in the context of a specific projects – consists in determining a subset $SA_{opt} \subseteq SA$ that contains one or more evaluation methods most suitable for the comparison task. Optimality is determined by (i) a qualitative assessment that reveals similarities and differences among the considered methods, and (ii) a quantitative assessment of each method's applicability with respect to *w* and *ref*.

2 REVIEW OF THE LITERATURE

Unfortunately, no techniques were found that specifically address the evaluation of the applicability of SA evaluation methods within the context of CQRS with ES architectural variations. However, several approaches exist for comparing these methods in general.

One such approach is the framework for classifying and comparing software architecture evaluation methods proposed in [5], which was developed by identifying similarities and differences among existing evaluation methods. The framework introduces a set of guiding questions designed to characterize SA evaluation methods. These questions include both general aspects, such as the number of quality attributes (QAs) considered and the estimated man-days required for applying the method, as well as more specific ones, such as whether the method provides support or guidance on non-technical aspects (e.g., social, organizational, managerial, or business issues) involved in the evaluation process. The proposed classification parameters (questions) were validated in [14] through expert surveys.

An extension to this framework was later published in [6], which arranged each element within four components, according to the Normative Information Model-based System Analysis and Design (NIMSAD) [15] evaluation framework. Three new dimensions were also added: Input and Output Management, Application Domain, and Stakeholder Benefits.

The output of applying this framework is a summary table describing the key characteristics of various SA evaluation methods. Although this helps narrow down the selection space and reduces the need to deeply study each method individually, the comparison remains qualitative and does not assess how well a particular method applies to a specific team and project.

An alternative comparison methodology is found in [7–8], which presents a family of experiments comparing Quality-Driven Architecture Derivation and Improvement [16] and ATAM [12]. The comparison was based on six parameters. Two of them are measurable: total time spent applying the method and effectiveness, which indicates how close the participant came to selecting the optimal architecture for the given project. A third parameter, efficiency, is calculated as the ratio of effectiveness to the application time. The remaining three parameters were drawn from the Technology Acceptance Model: Perceived Ease of Use, Perceived Usefulness, and Intention to Use. These were assessed through a post-experiment Likert-scale questionnaire containing a set of closed questions for each variable.

Among these parameters, effectiveness is perhaps the most interesting metric in the context of objectivity. It reflects how accurately a participant with relatively low qualifications can select an appropriate architectural solution using a given evaluation method. It was computed using the Euclidean distance between the n-dimensional vector of Non-Functional Requirements (NFR) values attained by the architecture selected by the participant and the optimal vector of values that could be achieved. Unfortunately, the authors did not provide details on the actual vector's values or how they were derived.

While this approach provides quantitative metrics for comparing two methods, it still does not answer the question of which method is best suited for a particular case. Moreover, the use of unqualified participants and a limited set of projects raise concerns about objectivity. Changes in participants, project characteristics, or NFRs could significantly impact the results of such experiments.

3 MATERIALS AND METHODS

Based on a systematic literature review [10–11], the most cited [3] scenario-based methods for SA design-time evaluation were selected: SAAM, ATAM, Cost-Benefit Analysis Method (CBAM), Architecture-Level Modifiability Analysis (ALMA), Systematic Quantitative Analysis of Scenarios' Heu-ristics (SQUASH) and Performance Assessment of Software Architecture (PASA).

This section is structured as follows:

– A proposal of an SA variation-oriented approach for assessing the applicability of selected methods to compare CQRS with ES architectural variations.

– A short overview of these methods, including the DSAV-CQRSES approach.

As a basis to compare SA variation evaluation methods, an enhanced version of the framework for classifying and comparing software architecture evaluation methods [6], modified concerning the NIMSAD evaluation framework.

However, the original framework is unsuitable for comparing SA evaluation methods in the context of their applicability to architectural variations, specifically, variations of CQRS with ES, for several reasons. First, the original set of questions does not accommodate the particular characteristics required for evaluating architectural variations. Secondly, it relies on qualitative rather than quantitative assessment, which delegates considerable analytical effort to the user.

Let us first define the terms SA and SA variation in order to clarify the distinction between them.

© Hruzin D. L., Lytvynov O. A., 2026
DOI 10.15588/1607-3274-2026-1-10

SA definition. Software architecture refers to the fundamental structure of a software system, encompassing its components, their relationships, and the principles guiding its design. In this context, multiple definitions exist. For example:

Lloyd and Galambos [17] define reference architectures as domain-specific architectural templates that aim to address the architectural concerns for a particular class of problems.

Bass et al. [18]: "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." Similar to the previous definition, this highlights that architecture can be understood as a metamodel [19].

Clements et al. [20]: "Software architecture is the set of structures needed to reason about a software system and the discipline of creating such structures and systems. Each structure comprises software elements, relations among them, and properties of both elements and relations." This definition conceptualizes architecture at a deeper level of abstraction – the model level.

In [21], software architecture is considered from several perspectives. On one hand, it is viewed as a set of basic concepts and constraints within which application functionality is to be specified and integrated. On the other hand, it serves as a means for addressing technical issues and quality requirements, as well as for assessing application functionality. As a result, Fritz Solms defines SA as the software infrastructure within which software components that address functional requirements of the software system can be specified, deployed, and executed.

In the presentation [22], Dr. Jean-Claude Franchitti provides several definitions, among which the following can be regarded as operating at a higher level of abstraction while also being the most comprehensive: "A set of artifacts (that is: principles, guidelines, policies, models, standards, and processes) and the relationships between these artifacts, that guide the selection, creation, and implementation of solutions aligned with business goals".

Based on the definitions above, SA is defined in the context of comparing SA evaluation methods as follows.

Definition: Software architecture is a structured set of artifacts (that is: principles, guidelines, policies, models, standards, and processes) and the relationships between these artifacts, established to enable a software system to satisfy functional and non-functional business requirements. It is characterized by the extent to which a system instantiated upon it fulfills specified business requirements under defined constraints.

SA variation definition. SA solutions are typically designed for a broad class of software systems and therefore offer the most universally applicable practices. Given the increasing need for flexibility in applications and systems, a trend has emerged toward flexible architectures [23] that can adapt to changes in business requirements and advancing technology stacks. The evolutionary architecture approach [24] assumes that SA must be continuously tested and adapted to produce more effective solutions, while ensuring that such evolution does not compromise key architectural concerns. As SAs are applied to various projects within a company, they give rise to a family of architectural variations that differ in complexity, performance, development time, and the level of developer expertise required. These differences have an impact on the development cost and maintainability of the software application.

In some sense, variations may be considered as deviations from the original SA. However, unlike those discussed in [25], they are aimed at improving the software product and optimizing the development process. Variations emerge as responses to specific technical challenges (e.g., event replay performance issues during aggregate reconstruction, complexity of event versioning, etc.) and as methods for reducing development and maintenance complexity.

SA variations, like architecture itself, are described by a metamodel; however, in the case of variations, the metamodel exhibits a higher degree of precision. From the perspective of Evolutionary Architecture, a set of architectural variations can be regarded as different stages of architectural evolution.

Thus, the following definition of an SA variation can be provided.

Definition: Software Architecture Variation is a purposeful deviation from a reference SA, characterized by the modification of one or more artifacts of this SA (that is: principles, guidelines, policies, models, standards, and processes) or the relationships between these elements. It is strategically implemented to optimize a software system's ability to fulfil both functional and non-functional business requirements with maximum efficiency, as measured by current and projected resource expenditure, development effort, and maintenance complexity.

Typically, SA variations emerge and evolve within bounded organizational contexts (e.g., enterprises or development communities) in response to specific technical challenges, operational constraints, or evolving business imperatives of projects under development. Thus, the essence of a SA variation lies in the modification of one or more artifacts of the architecture (i.e., principles, guidelines, policies, models, standards, and processes) or in a deviation from them. However, such deviations must not involve abandoning the fundamental principles of the architecture. For example, rejecting the segregation of commands and queries within the CQRS approach would constitute a departure from the architectural paradigm itself and therefore cannot be regarded as a variation. Similarly, discarding the event store and event-based operations in Event Sourcing also violates core principles. In contrast, using an event store that no longer serves as the source of truth may be considered a deviation from the original architecture and thus qualify as a variation. Ultimately, the determination of whether a modification constitutes a new architectural solution or a variation of an existing one should remain within the discretion of the modification creator.

DOI 10.15588/1607-3274-2026-1-10

For the majority of SA evaluation methods, the QA set commonly used in SA evaluation includes characteristics such as usability, security, reliability, portability, cost, and others. These attributes are also formalized in ISO/IEC 25002:2024 [26]. While all of these parameters are relevant to SA evaluation, the nature of CQRS with ES architectural variations places primary emphasis on maintainability and performance. Within the context of CQRS with ES, other QAs are largely equivalent across the majority of variations [4] and are therefore not the focus of differentiation.

Taking into account the specific characteristics of architectural variations, the questions proposed by the original framework for classifying and comparing SA evaluation methods should be revised.

It is a good point for the method if it includes not only an SA definition but also a definition of SA variation. That will help the team to understand the specificity of architectural candidates. The next objective of the framework is to clarify whether the method's specific goals include the comparison of architectural variations. QAs should cover those that are relevant to the user's needs for comparing candidates within the context of their application (i.e., SA variation-oriented QAs).

As an output, most methods produce documentation describing the architectural candidates. In terms of output quality, the following aspects are evaluated:

– Whether the documentation is ready for direct use during the implementation phase or further elaboration is required.

– Whether trade-offs between architectural alternatives are explicitly addressed.

– Whether the method highlights a recommended architecture or just provides information to support further decision-making.

Different methods offer various benefits. For architectural variation comparison, the most valuable ones include the identification of the optimal solution and an approximate estimate of its implementation effort.

One of the most challenging aspects of applying SA evaluation methods is involving business stakeholders in evaluating QAs. It is often difficult to explain technical QAs (such as modifiability or portability) to non-technical team members. Therefore, it is a great point if stakeholder participation is minimized, and they are asked to assess only easily understandable criteria (e.g., estimated development time in man-hours, or system response time in milliseconds). The usability of a method is also affected by the required resources, such as team size and the time investment. Spending several man-weeks to evaluate each architectural solution can be unacceptably expensive.

Methods provide different forms of architectural candidate description. Formal modeling languages such as Unified Modeling Language [27] or other Architecture Design Languages (ADL) [28] (e.g., module or logical views) are often recommended. However, the depth of these descriptions also differs: some methods provide only a high-level overview, while others model architecture with greater precision. For distinguishing between structurally similar SA variations, especially when relying on expert judgment, clear, formal, and sufficiently detailed descriptions are essential. This leads to the question: Does the method provide a formal and detailed description of the architectural candidates?

Even when detailed descriptions are used, if a method relies on expert judgment, it introduces a human factor, making the evaluation subject to uncertainty. Therefore, it is important to clearly distinguish which parts of the evaluation are based on expert opinions and which are grounded in experiments or statistical data. Although evaluation approaches that rely on empirical experiments and statistical analysis may offer increased objectivity, they are still subject to uncertainty. Thus, it is essential to consider the extent to which a method is affected by uncertainty and whether it includes mechanisms for uncertainty mitigation.

Finally, regarding the validation of a method, it is necessary to clarify whether the method has been validated specifically in the context of architectural variations or not.

The components and attributes of the framework and the evaluation questions are presented in Table 1.

Table 1 – The components and attributes of the framework and the evaluation questions

Component	Elements	Original explanation	Variation-oriented explanation
Context	SA definition	Does the method explicitly consider a particular definition of SA?	How closely do the method's definitions of SA and SA variation align with those considered in this paper?
	Specific goal	What is the particular goal of the methods?	Is the selection between structurally similar SA alternatives addressed by the method's objectives?
	Quality attributes	How many and which quality attributes are covered by the method?	Are variation-specific QAs (e.g., complexity and performance) covered by the method?
	Applicable stage	Which is the most appropriate development phase to apply the method?	Is the method applicable at the design stage of system development?
	Input & output	What are the inputs required and outputs produced?	To what extent is the resulting technical documentation ready for direct use during implementation? Does the method facilitate trade-off analysis? Does the method identify a recommended architecture, or does it only provide additional information about the alternatives?
	Application domain	What is/are the application domain(s) the method is mostly applied?	Is the method validated or considered applicable in domains similar to the target system?

Continuation of Table 1

Stakeholders	Benefits	What are the benefits of the method to the stakeholders?	Does the method support the selection of the optimal solution? Does the method provide an approximate estimate of the effort required for system implementation?
	Involved Stakeholders	Which groups of stakeholders are required to participate in the evaluation?	Does the method require involvement of multiple stakeholders? How many actions or inputs are expected from stakeholders?
	Process support	How much support is provided by the method to perform various activities?	How much support is provided by the method to perform various activities?
	Socio-technical issues	How does method handle non-technical (e.g. social, organisational issues)?	Does the method address non-technical issues (e.g., social and organizational factors)?
	Required resources	How many man-days are required? What is the size of evaluation team?	How long does the application of the method take? What is the typical team size required to apply the method? How demanding are the architectural skills required from the users of the method?
Contents	Method's activities	What are the activities to be performed and in which order to achieve the goals?	What are the activities to be performed and in what order to achieve the goals?
	SA description	What form of SA description is recommended (e.g., formal, informal, particular ADL, views etc.)?	Does the method provide a formal and detailed description of the architectural candidates?
	Evaluation approaches	What types of evaluation approaches are used by the method?	To what extent is the method subject to uncertainty? Are there any techniques applied to mitigate uncertainty? How much of the evaluation is grounded in quantitative metrics and experimental data versus expert assessment?
	Tool support	Are there tools or experience repository to support the method and its artefacts?	Does the method offer tools to facilitate or partially automate its use?
Reliability	Maturity of method	What is the level of maturity (inception, development, refinement or dormant)?	What is the level of maturity (inception, development, refinement or dormant)?
	Method's validation	Has the method been validated? How has it been validated?	Has the method been validated? Has the method been applied specifically to architectural variations?

The modified framework highlights key points relevant for comparing architectural variations; however, the comparison remains qualitative. To obtain quantitative indicators of the effectiveness of SA evaluation methods, the following algorithm is proposed:

1) Evaluate each criterion by answering the framework's questions for each method using a Yes, Yes/No, No scale (Y, Y/N, N) for binary-type questions, and a Low, Medium, High scale (L, M, H) for degree-based questions. Note: This point applies to all criteria except "Method's activities". While it is important for qualitative comparison, it does not influence the decision regarding the selection of an SA evaluation method and should not be relied upon in quantitative evaluation.

2) Assign weights to each criterion based on its impact on the effectiveness of the method's application. It is proposed to determine weights through expert judgment.

It is important to note that the results of the first two steps can be reused in subsequent applications of the method.

3) Determine quantitative equivalents for the values H, M, L and Y, Y/N, N.

4) Define a reference (etalon) method, representing the characteristics for the ideal candidate, the team would desire to use, and the weights (priorities) assigned to each question. To do this, the team answers the framework's questions, specifying the responses and setting the priority that are ideal for their context. This process is suggested to be conducted through structured collective discussion and subsequent voting.

5) The formalized responses to the framework's questions constitute a multidimensional vector for each candidate method and the reference method. The values of the vector's elements are replaced with their quantitative equivalents.

$$p=(Y,N,L,H), iff \quad Y=H=3, N=L=1$$
$$then \quad p=(3,1,1,3).$$

6) The deviation of each candidate method from the reference method is measured using the Euclidean distance metric:

$$D(p,q)=\sqrt{\sum_{i=1}^{m} w_i \cdot (p_i-q_i)^2}, i\in[1,m],$$

where w_i represents the weight of the i-th criterion, q is the vector representing the reference method, p is the vector representing the candidate method, m is the number of criteria.

7) Based on the computed distances, the effectiveness of each method's applicability is derived using the formula (1):

$$Effectiveness(p)=1-\frac{D(p,ref)}{\text{Max}}, \tag{1}$$

where *ref* is the vector representing the reference method, p is the vector representing the candidate method, *Max* is the maximum possible distance from the etalon.

8) As a result, a vector of effectiveness values for the SA evaluation methods is obtained:

$$E=\langle e_1,e_2,...,e_n\rangle, e_i=\chi(sa_i), sa_i\in SA, i\in[1,n],$$

DOI 10.15588/1607-3274-2026-1-10

where $\chi : SA \to E$ and $\chi' : E \to SA$ are transformation functions, SA is the set of SA evaluation methods, e_i is the effectiveness of the i-th method, n is the number of SA evaluation methods considered in the comparison.

9) Next, by sorting this array, we obtain an ordered vector E_{asc}.

$$E_{asc} = \sigma(E) = \langle ea_1, ea_2, ..., ea_n \rangle, \tag{2}$$

where $ea_1 \le ea_2 \le \ldots \le ea_n$, $\sigma : E \to E_{asc}$ is the sorting function by ascending value and $\sigma' : E_{asc} \to E$ is an inverse function of σ.

10) Determining a vector $E_{\max}$, consisting of elements whose values are close to the maximum value in vector E_{asc} (ea_n). In most cases, this set will consist of a single element:

$$E_{\max} = \max_j (ea_j) = \langle ea_n \rangle. \tag{3}$$

However, when the difference between ea_n and ea_{n-1} is not significant, the decision-maker should consider several options. In this case, the vector $E_{\max}$ will include multiple elements (Fig. 1).

$$E_{\max} = \{ ea \in E_{asc} \mid d = ea_n - ea_{n-k}, d < th, k \in \mathbb{N} \}, \tag{4}$$

where th is the value that defines the maximum allowable deviation from the maximum value at which ea_j is still considered as one of the most effective values.

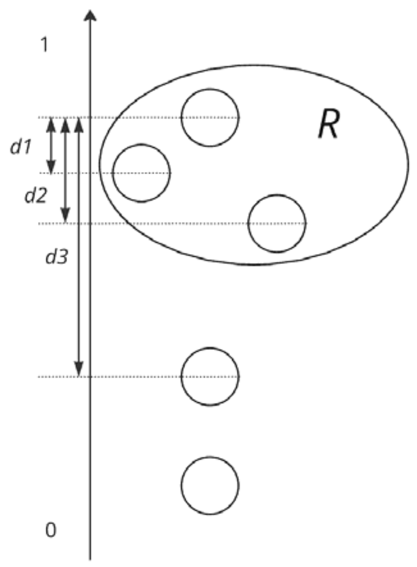

Figure 1 – Set of most suitable results $E_{\max}$. $d1$, $d2 < th$, $d3 > th$

11) The output of the algorithm is a set $SA_{opt} \subseteq SA$, that contains the most suitable methods deriving from $E_{\max}$:

$$SA_{opt} = \chi'(\sigma'(E_{\max})). \tag{5}$$

The initial step in comparing SA evaluation methods is the description of each method:

Software Architecture Analysis Method [13] is one of the earliest methods proposed for software architecture evaluation. Initially, it focused primarily on maintainability QA. However, subsequent extensions of the method have incorporated additional concerns such as reusability [29], flexibility and complexity [30], and so forth.

Prior to applying the method, the architectural alternatives under evaluation must be described in detail. The SAAM process consists of five main steps: i. characterize a canonical functional partitioning for the domain; ii. map the functional partitioning onto the structural decomposition of the architecture; iii. select a set of QAs for evaluation; iv. identify a set of concrete tasks which test the desired QAs; v. evaluate the degree to which each architecture provides support for each task.

While SAAM is primarily a qualitative evaluation method, one of its distinctive features is the use of small-scale experiments and benchmarks. For example, a benchmark might involve measuring the time or effort required to add a new option to a user's menu bar, reflecting some piece of application functionality. Based on qualitative assessments and experimental benchmarks of application modification, evaluators analyse alternatives and determine whether the candidate provides architectural support in the context of the QA.

The outcome of applying SAAM includes a report that documents the architectural alternatives and their qualitative evaluation of the project's quality requirements, as well as the experimental applications and modification benchmarks developed during the assessment.

The duration of the evaluation depends on the number of alternatives and the complexity of the experimental tasks. On average, conducting a SAAM assessment requires approximately three days for a team of four evaluators, in addition to stakeholder participation and preparatory activities [5].

Limitations of the method include a strong reliance on expert judgment, which introduces a human factor, as well as the lack of quantitative metrics for assessing QAs. Furthermore, SAAM does not explicitly address trade-offs between conflicting QAs.

Architectural Trade-off Analysis Method [12] is a technique for analysing SA that provides a well-defined sequence of evaluation steps, including a schedule and clearly defined roles for each participant. The application of ATAM involves a preparation phase, typically handled by the lead architect or architecture team, and two evaluation phases.

The first phase of ATAM is architecture-centric. It begins with a presentation of the ATAM to the evaluation team, followed by the analysis of the problem domain and identification of business goals, specification of requirements for architectural solutions, and the selection of relevant QAs, which can be different (e.g., availability, security, usability, and modifiability). A Quality Attribute Utility Tree is then generated.

Each architectural alternative is then analysed with a focus on four aspects: understanding the approach, identifying well-known weaknesses, recognizing sensitivity points, and finding interactions and trade-offs with other approaches. For each alternative, both approach-specific and quality-attribute-specific questions are discussed, and risks associated with the identified concerns are documented. The outcomes of this phase include: a document with the system's business requirements, a Quality Attribute Utility Tree, and a document outlining the architectural alternatives along with the associated risks and concerns.

The second phase is stakeholder-centric and concentrates on eliciting stakeholder points of view and verifying the results of the first phase. It involves a brainstorming session in which a wide range of scenarios are generated. The scenarios cover both system use cases and possible future modifications. These scenarios are then prioritized via stakeholder voting. Based on these prioritized scenarios, each architectural alternative is analysed again. Any sensitivity or trade-off point is treated as a potential risk.

The final deliverables of ATAM include the documented architectural approaches, the set of scenarios and their prioritization, the collection of attribute-based questions, the utility tree, the list of identified risks and non-risks, as well as the discovered sensitivity and trade-off points. Based on this information, the team selects the architectural approach that demonstrates the lowest level of risk. If none of the evaluated alternatives is deemed sufficiently suitable, the process iterates with the preparation of additional architectural candidates.

Successful application of ATAM requires a multidisciplinary team. A minimum of three evaluators and a representative set of stakeholders is recommended. The core evaluation spans two full working days, besides preparation time. ATAM does not provide formal tool support, although certain steps could potentially be optimized with modern technologies such as large language models. Like most scenario-based approaches, the quality of the evaluation outcome is strongly dependent on the expertise of the participants.

Cost-Benefit Analysis Method [31] is an architecture evaluation technique that extends the ATAM framework by incorporating cost-benefit analysis of architectural design decisions. The prerequisites for applying CBAM include a prior ATAM assessment and the availability of a cost estimation model for implementing architectural alternatives.

The CBAM process involves several steps. Initially, stakeholders assess the benefits of QAs by assigning them relative weights and ranking each alternative based on its contribution (Cont) to each QA on a scale from –1 to 1. This is typically done through stakeholder voting. The benefit of each architectural strategy is then computed using the following formula:

$$Benefit(AS_i) = \sum_j (Cont_{ij} \times QAscore_j).$$

In the next step, the expected cost of each alternative is estimated. If precise data (e.g., statistical data or specific implementation prices) is available, it is used directly; otherwise, it is suggested to estimate the cost using qualitative scales such as High, Medium, or Low. The last step before making a decision is the calculation of the Return on Investment (ROI) metric for each alternative using the following formula:

$$ROI(AS_i) = Benefit(AS_i) / Cost(AS_i).$$

This method heavily depends on expert judgment, particularly due to the abstract nature of some QAs (e.g., reliability or modifiability), which may be interpreted inconsistently across team members, leading to uncertainty in results. Because of this, in the second version of CBAM [32], QAs are replaced with specific scenarios for each QA, and additional evaluation iterations are introduced. As a result, the assessment yields not a single benefit value per architectural alternative, but a set of values. The minimum and maximum values in this set are interpreted as the boundaries of a confidence interval, reflecting the uncertainty inherent in the expert evaluations.

The application of CBAM requires a team comparable in size to that used for ATAM and typically takes approximately two additional working days. The outcomes of this method include the results obtained through ATAM (utility tree, scenarios, risks), ROI estimates for each alternative, and a confidence matrix representing the degree of certainty in the ranking under uncertainty conditions.

Architecture-Level Modifiability Analysis [33] is a method for SA evaluation with a focus on modifiability (e.g., maintenance cost prediction and risk assessment). The evaluation flow consists of five main steps: determine the aim of the analysis, describe the software architecture, find the set of relevant scenarios, determine the effect of the scenarios, and conclude the analysis.

During the ALMA application, the evaluator sets one or more analysis objectives. The first objective is to estimate the cost of maintenance or modification of the application. The second is to identify potential changes for which the architecture is inflexible and to assess the associated risks. The third objective is to support architectural decision-making by comparing multiple SA candidates and selecting the most suitable one.

The method for comparing candidates focuses on finding extreme scenarios that stress the architecture and evaluating how each candidate responds. The scoring process is left to the discretion of the evaluators. This may involve comparing quantitative metrics, if available, or aggregating expert judgment ratings across all scenarios.

DOI 10.15588/1607-3274-2026-1-10

To improve result accuracy, the method recommends conducting the evaluation in several iterations. During each iteration, the team discusses a set of questions (or experiments) for each candidate architecture and assigns ratings for every scenario.

The result of applying the ALMA method is an assessment of the architecture candidates in terms of maintainability and reusability. The data is presented in a table that, for each candidate, indicates which application components need to be modified or added, along with the affected requirements. The method does not provide mechanisms for handling trade-offs between maintainability and other QAs.

Systematic Quantitative Analysis of Scenarios' Heuristics [34] is a method that focuses on the quantitative evaluation of both architectural alternatives and stakeholder-defined objectives to reduce uncertainty. Its application includes identifying stakeholders and objectives, making objectives quantifiable, analysing and aggregating scenarios, improving scenarios, and selecting the optimal options. To enhance precision, the analysis process is conducted in multiple iterations.

In [34], QAs such as usability and performance are used as evaluation criteria. For each QA, a set of objectives is formulated in the form of refined system requirements. Stakeholders assign quantitative values to each objective across five levels (ranging from Minimal to Excellent). Each level is associated with a distinct colour.

Subsequently, SA candidates are analysed in the context of the defined objectives, taking into account detailed technical aspects. Where metric-based evaluation is not feasible, it is suggested to use expert judgment with a Low / Medium / High scale.

The output of the method is presented as a table, where the columns represent SA candidates and the rows represent objectives, grouped by QAs. Each cell contains the evaluation of an SA candidate for the given objective and is color-coded according to the corresponding level.

A key limitation of the method is its complexity in application. When evaluation is tailored to a non-typical task, a considerable number of objectives can lead to high preparation and experimentation effort, which makes the method difficult to apply in practice.

Performance Assessment of Software Architecture [35] is a method focused on achieving performance objectives. Its goal is either to improve the performance of an existing system or to select an SA that meets the performance requirements of a new system.

The PASA application flow consists of 10 steps. The first step is an overview of the reasons for conducting an architectural assessment and the assessment process itself, presented to the entire team, including developers and stakeholders. Then, documentation is prepared for the architecture of the existing or planned application (if not already available), and an overview of the architecture is presented to the team. In the following steps, critical use cases affecting responsiveness and scalability, as well as key performance scenarios, are identified. These scenarios are documented and typically represented using UML.

Next, the team identifies the performance objectives. Based on the presented data, the participants conduct a more detailed discussion of the architecture and its specific features that influence key performance scenarios. If the analysis reveals performance issues, architectural alternatives that meet the performance requirements are proposed. The results and recommendations are then presented to the entire team, and an economic analysis of the costs and benefits of the proposed solution is conducted.

Performance improvement methods include practices such as identifying deviations from the current architectural style, detecting performance-related antipatterns [36], proposing alternative interactions between components, etc. Using software performance engineering techniques [37], evaluators perform performance modeling, including best- and worst-case analyses to address uncertainty.

The assessment process usually takes 1–2 weeks. The outputs of the method include:

– Documentation of the current architecture and main processes (if not already available).

– A set of architectural alternatives that meet performance requirements, with recommendations for selecting among them.

– A rough comparison of the costs of analysis and subsequent improvements versus the potential costs of additional hardware and development effort that would have been required if the problems had not been detected in time.

The main drawbacks of the method are the lack of trade-off analysis with other QAs and the focus only on critical use cases. If a large number of scenarios need to be evaluated, the analysis may become time-consuming.

Information Technology for Decision-making Support regarding CQRS with ES Architectural Variations [4] is a method designed to evaluate evolutionary architectural variations within a single SA style. Its primary focus is on comparing different variants of CQRS with ES architecture. Given the nature of these variations, the central QAs under consideration are maintainability and performance. Other QAs are largely equivalent across the majority of variations and are therefore not the focus of differentiation, at least for CQRS with ES architecture. DSAV-CQRSES takes into account detailed technical aspects, but unlike SQUASH, it introduces a classification of typical use cases, which significantly reduces the effort required to conduct the experiments.

The method supports three distinct evaluation objectives: assessing implementation/modification complexity, supporting architectural decision-making by selecting the most suitable variation, and evaluating migration complexity between variations.

During the preparation phase for selecting an architectural variation, requirements are gathered, a preliminary estimation of the number of use cases of various types is calculated, and the expected proportion of addition new / modification old use cases after the release of the minimum viable product phase is assessed.

© Hruzin D. L., Lytvynov O. A., 2026
DOI 10.15588/1607-3274-2026-1-10

The support architectural decision-making by selecting the most suitable SA variation algorithm looks as follows:

1. Identification of QAs.
2. Informal description of the considered architectural variations.
3. Estimation of implementation/modification complexity for each variation:
    3.1. Formalization of the architectural variations:
        3.1.1. Classification of use case types;
        3.1.2. Identification of processes corresponding to each use case class;
        3.1.3. Formal process modelling using the activity model.
    3.2. Evaluation of the processes.
    3.3. Evaluation of the architectural variation.
4. Identification of metrics requiring a Representative Test Project (RTP):
    4.1. Definition of the RTP.
    4.2. Implementation of the RTP.
    4.3. Metric collection based on the RTP.
    4.4. Aggregation and preparation of results for applying a Multi-Criteria Decision Analysis (MCDA) method (automated).
5. Application of the MCDA method, such as AHP (automated).

The method can be used to achieve each objective independently. Its primary advantage is the minimal reliance on expert judgment. While human input is needed during the process modelling phase, high-level architecture expertise is not required to construct use case and process diagrams. Due to the automation of calculations, stakeholders do not need to predefine QA priorities; instead, they can visually analyse the data to identify the most suitable variation under different conditions and select an appropriate trade-off.

The output includes the visualized applicability metrics of each architectural variation under different QA values (in percentages), as well as documentation describing the processes.

It is important to note that the time required to apply the method increases with the number of use case classes. The method is specifically designed for evaluating CQRS with ES architecture, which typically involves two to three use case classes, depending on stakeholder needs. For applications with a large number of unique processes, the method may become less applicable due to the associated effort.

4 EXPERIMENTS

The experiment involves comparing a set of scenario-based methods for SA design-time evaluation: SAAM, ATAM, CBAM, ALMA, SQUASH, PASA and DSAV-CQRSES. It consists of three stages.

1) The methods are qualitatively compared using the modified comparison framework.

2) The qualitative evaluation is translated into a quantitative form, and criterion weights are applied. As a result, each method is represented by a multidimensional vector corresponding to its evaluation on each criterion.

3) A reference method is then introduced, reflecting the characteristics desired by the DBB Software team for evaluating CQRS with ES architectural variations. The effectiveness of each method is calculated using formula (1), after which the methods are ranked, and the top candidates are selected, using formulas (2–4).

5 RESULTS

The results of each stage are summarized in Tables 2–3.

Table 2 – Results of the qualitative comparison of SA evaluation methods

Criteria	SAAM	ATAM	CBAM	ALMA	SQUASH	PASA	DSAV-CQRSES
How closely do the method's definitions of SA and SA variation align with those considered in this paper?	SA definition is left to users.	SA definition is left to users.	SA definition is left to users.	Not provided.	Not provided.	Not provided.	Specified definition for SA variations.
Is the selection between structurally similar SA alternatives addressed by the method's objectives?	No.	No.	No.	No.	No.	Different solutions applied to the same architecture are considered to address performance issues.	Yes. Method is developed for SA variations comparison.
Are variation-specific QAs (e.g., complexity and performance) covered by the method?	Just modifiability (complexity).	Yes.	Yes.	Just modifiability (complexity).	Yes.	Just performance.	Yes.

Continuation of Table 2

Is the method applicable at the design stage of system development?	Yes.	Yes.	Yes.	Yes.	Yes.	Yes.	Yes.
To what extent is the resulting technical documentation ready for direct use during implementation?	Informal description of candidates, list of scenarios, experimental docs and benchmarks, if available.	Informal description of candidates, list of scenarios and identified risks.	Informal description of candidates, list of scenarios and identified risks.	Informal description of candidates.	Informal description of candidates, experimental docs, if available.	Informal description of candidates and critical processes.	Informal description of candidates, formal specification of their typical processes.
Does the method facilitate trade-off analysis?	No.	Yes, manual.	Yes, manual.	No.	Preparations for trade-off. Left to users.	Between performance and costs, manual.	Between performance and costs, automatic.
Does the method identify a recommended architecture, or does it only provide additional information about the alternatives?	Evaluates the modifiability parameters of the candidates. Left to users.	Identifies risks across multiple QAs. Left to users.	Identifies risks across multiple QAs. Provides a cost/benefit assessment. Left to users.	Evaluates the modifiability parameters of the candidates. Left to users.	Evaluates multiple parameters of the candidates. Left to users.	The selection of recommended options is performed manually by the evaluator.	Yes, based on performance and complexity parameters.
Is the method validated or considered applicable in domains similar to the target system?	Various domains, including medical.	Various domains, including medical.	Various domains, including medical.	Various domains, including medical.	Verified on medical domain.	Various domains.	Domains compatible with CQRS with ES, including medical.
Does the method support the selection of the optimal solution?	Yes.	Yes.	Yes.	Yes.	Yes.	Yes.	Yes.
Does the method provide an approximate estimate of the effort required for system implementation?	Yes, based on benchmarks.	No.	Not explicitly, based on costs evaluation.	Not explicitly, based on components to be changed.	Not explicitly, based on experiments.	No.	Yes, based on experiments.
Does the method require involvement of multiple stakeholders?	Yes.	Yes.	Yes.	Yes.	No, only for discussion and clarification of requirements.	No, only for discussion of requirements and scenarios.	No, only for discussion of requirements.
How many actions or inputs are expected from stakeholders?	Participate in expert evaluation of scenarios.	Participate in expert evaluation of scenarios.	Participate in expert evaluation of scenarios.	Participate in expert evaluation of scenarios.	Formulate refined requirements.	Formulate requirements and critical scenarios.	Formulate requirements.
How much support is provided by the method to perform various activities?	Not explicitly addressed.	Comprehensively covered.	Comprehensively covered.	Embedded in method description.	Embedded in method description.	Embedded in method description.	Embedded in method description.
Does the method address non-technical issues (e.g., social and organizational factors)?	Such issues briefly mentioned.	Sufficiently provided.	Sufficiently provided.	Not explicitly addressed.	Not explicitly addressed.	Not explicitly addressed.	Not addressed. Minimized, due to reduced communication with stakeholders.
How long does the application of the method take?	Apart from initial & post preparation, 3 days.	Apart from initial & post preparation, 2 days.	Apart from initial & post preparation, 2 days (ATAM) and 3 days.	Not specified.	The method requires a considerable amount of time to collect all the required data.	1–2 weeks.	Apart from initial & post preparation, 2 days.

Continuation of Table 2

What is the typical team size required to apply the method?	4-person evaluation team & stakeholders.	3-person evaluation team & stakeholders.	3-person evaluation team & stakeholders.	Not specified.	Not specified.	Not specified.	2-person evaluation team & stakeholders.
How demanding are the architectural skills required from the users of the method?	High, for the expert judgment and experiments.	High, for the expert judgment.	High, for the expert judgment.	High, for the expert judgment.	High, for the expert judgment and experiments.	High, for the performance modeling and generating alternatives.	Moderate, for experiments and use case documentation.
What are the activities to be performed and in what order to achieve the goals?	6 activities.	9 activities in 2 phases.	9 activities in 2 phases (ATAM) & 4 activities performed over several iterations.	5 activities carried out sequentially.	4 activities & 3 activities performed over several iterations.	10 activities.	4 activities & 6 activities performed over several iterations.
Does the method provide a formal and detailed description of the architectural candidates?	High-level description using ADL; focuses on identifying maintainability-related risks.	High-level description using ADL; emphasizes risks related to multiple QAs.	High-level description using ADL; focuses on economic trade-offs between QAs.	Supports formal description for complex architectures; emphasizes maintainability-related risks.	High-level description using ADL; experiments partially cover system behavior and processes.	Provides detailed description and performance modeling of critical processes and components.	Offers a formal description of SA processes, using ADL and activity models.
To what extent is the method subject to uncertainty?	Highly, due to its reliance on expert judgment.	Highly, due to its reliance on expert judgment.	Highly, due to its reliance on expert judgment.	Highly, due to its reliance on expert judgment.	Moderate, due to partial reliance on expert judgment and remaining variability in the experiments.	Moderate, due to variability in the modeling.	Moderate, due to possible inaccuracies in the source data (use cases) and remaining variability in the experiments.
Are there any techniques applied to mitigate uncertainty?	Implicit.	Implicit, extra iterations of architecture analysis.	Yes, In V2 extra iterations and value boundaries are added.	Implicit.	Implicit, within experiments.	Implicit, within performance modeling.	Implicit, within experiments.
How much of the evaluation is grounded in quantitative metrics and experimental data versus expert assessment?	Primarily expert judgement.	Primarily expert judgement.	Suggested to use statistics or real prices for costs if possible.	Primarily expert judgement. Left to users.	Primarily metrics and experiments.	Primarily metrics and experiments.	Primarily metrics and experiments.
Does the method offer tools to facilitate or partially automate its use?	Partially available.	Not available.	Not available.	Not available.	Not available.	Not available.	Partially automated.
What is the level of maturity (inception, development, refinement or dormant)?	Dormant.	Dormant.	Dormant.	Dormant.	Refinement.	Refinement.	Development.
Has the method been validated?	Yes.	Yes.	Yes.	Yes.	Found a couple articles with its application.	Found a couple articles with its application.	For a couple of projects.
Has the method been applied specifically to architectural variations?	No.	No.	No.	No.	No.	No.	Yes.

A typical project, derived from the analysis of multiple DBB Software projects employing the CQRS with ES architecture, has been selected as the reference case. At this stage, a specific variation needs to be chosen. The software system under development belongs to the medical domain and requires a high level of maintainability and good performance.

Table 3 presents the answers from Table 2 converted into a more formalized form through expert evaluation, describes the reference case, and calculations of the effectiveness of the methods using formula (1).

DOI 10.15588/1607-3274-2026-1-10

Table 3 – Results of the quantitative comparison of SA evaluation methods

Criteria	Weight	SAAM	ATAM	CBAM	ALMA	SQUASH	PASA	DSAV-CQRSES	Ref	Max
How closely do the method's definitions of SA and SA variation align with those considered in this paper?	3	L	L	L	L	L	L	M	H	L
Is the selection between structurally similar SA alternatives addressed by the method's objectives?	5	N	N	N	N	Y/N	Y	Y	Y	N
Are variation-specific QAs (e.g., complexity and performance) covered by the method?	10	Y/N	Y	Y	Y/N	Y	Y/N	Y	Y	N
Is the method applicable at the design stage of system development?	10	Y	Y	Y	Y	Y	Y	Y	Y	N
To what extent is the resulting technical documentation ready for direct use during implementation?	1	Y/N	Y/N	Y/N	Y/N	N	Y/N	Y/N	Y/N	N
Does the method facilitate trade-off analysis?	7	N	Y	Y	N	Y/N	Y/N	Y/N	Y	N
Does the method identify a recommended architecture, or does it only provide additional information about the alternatives?	2	N	N	N	N	N	N	Y	Y	N
Is the method validated or considered applicable in domains similar to the target system?	8	Y	Y	Y	Y	Y	Y/N	Y	Y	N
Does the method support the selection of the optimal solution?	10	Y	Y	Y	Y	Y	Y	Y	Y	N
Does the method provide an approximate estimate of the effort required for system implementation?	4	Y/N	N	Y/N	Y/N	Y/N	Y/N	Y	Y	N
Does the method require involvement of multiple stakeholders?	4	Y	Y	Y	Y	Y/N	Y/N	N	N	Y
How many actions or inputs are expected from stakeholders?	7	H	H	H	H	M	M	L	L	H
How much support is provided by the method to perform various activities?	5	L	H	H	M	M	M	M	H	L
Does the method address non-technical issues (e.g., social and organizational factors)?	2	Y/N	Y	Y	Y/N	Y/N	Y/N	Y/N	Y/N	N
How long does the application of the method take?	5	M	M	H	M	H	H	M	M	L
What is the typical team size required to apply the method?	5	H	M	M	M	M	H	L	L	H
How demanding are the architectural skills required from the users of the method?	8	H	H	H	H	H	H	M	M	L
Does the method provide a formal and detailed description of the architectural candidates?	4	N	N	N	Y/N	N	Y/N	Y	Y	N
To what extent is the method subject to uncertainty?	4	H	H	H	H	M	M	M	L	H
Are there any techniques applied to mitigate uncertainty?	8	Y/N	Y/N	Y	Y/N	Y/N	Y/N	Y/N	Y	N
How much of the evaluation is grounded in quantitative metrics and experimental data versus expert assessment?	7	L	L	M	L	H	H	H	H	L
Does the method offer tools to facilitate or partially automate its use?	2	Y/N	N	N	N	N	N	Y	Y	N
What is the level of maturity (inception, development, refinement or dormant)?	6	H	H	H	H	M	M	L	H	L
Has the method been validated?	6	Y	Y	Y	Y	Y/N	Y/N	Y/N	Y	N
Has the method been applied specifically to architectural variations?	6	N	N	N	N	N	N	Y	Y	N
Euclidean distances		16.37	14.66	13.38	15.23	11.96	12.57	7.55		22.54
Effectiveness		0.27	0.35	0.41	0.32	0.47	0.44	0.67		

For simplicity in experimentation, the following numeric values are used in this study: L = 1, M = 2, H = 3; N = 1, Y/N = 2, Y = 3.

The weights for each parameter were determined based on the expert judgment of the DBB Software development team conducting the evaluation. They reflected the extent to which each factor influences the team's convenience in applying the method.

For example, the distance between the reference method and DSAV-CQRSES is calculated as follows:

Reference vector:

$$ref = (H, Y, Y, Y, Y/N, Y, Y, Y, Y, Y, N, L, H, Y/N, M, L, M, Y, L, Y, H, Y, H, Y, Y) =$$
$$= (3,3,3,3,2,3,3,3,3,3,1,1,3,2,2,1,2,3,1,3,3,3,3,3,3).$$

Maximum vector:

$$\max = (L, N, N, N, N, N, N, N, N, N, Y, H, L, N, L, H, L, N, H, N, L, N, L, N, N) =$$
$$= (1,1,1,1,1,1,1,1,1,1,3,3,1,1,1,3,1,1,3,1,1,1,1,1,1).$$

Weights vector:

$$w = (3,5,10,10,1,7,2,8,10,4,4,7,5,2,5,5,8,4,4,8,7,2,6,6,6).$$

DSAV-CQRSES vector:

$$p_{DSAV-CQRSES} = (M, Y, Y, Y, Y/N, Y/N, Y, Y, Y, Y, N, L, M, Y/N, M, L, M, Y, M, Y/N, H, Y, L, Y/N, Y) =$$
$$= (2,3,3,3,2,2,3,3,3,3,1,1,2,2,2,1,2,3,2,2,3,3,1,2,3).$$

Maximum distance to the Reference:

$$D(\max, ref) = \sqrt{\sum_i w_i \cdot (\max_i - ref_i)^2} = 22.54, i \in [1,25]$$

DSAV-CQRSES to the Reference distance:

$$D(p_{DSAV-CQRSES}, ref) = \sqrt{\sum_i w_i \cdot (p_{DSAV-CQRSES} - ref_i)^2} = 7.55, i \in [1,25].$$

DSAV-CQRSES effectiveness:

$$Effectiveness(p_{DSAV-CQRSES}) = 1 - \frac{D(p_{DSAV-CQRSES}, ref)}{D(\max, ref)} = 1 - \frac{7.55}{22.54} \approx 0.67.$$

Obtain an ordered vector E_{asc} using formula (2):

$$E_{asc} = \{\chi(sa_{SAAM}), \chi(sa_{ALMA}), \chi(sa_{ATAM}), \chi(sa_{CBAM}), \chi(sa_{PASA}), \chi(sa_{SQUASH}), \chi(sa_{DSAV-CQRSES})\}.$$

Determine the result set using formulas (3–5). The maximum allowable deviation (*th*) in formula (4) let us set to 0.05. With this threshold, the resulting set contains a single element:

$$SA_{opt} = \{sa_{DSAV-CQRSES}\}.$$

6 DISCUSSION

The existing approaches reviewed in the literature do not consider evaluation in the context of SA variations. Due to the specific nature and similarity of SA variations, most traditional SA evaluation methods are not well-suited for their comparison. The modified version of the framework for classifying and comparing SA evaluation methods, adapted to support the comparison of methods for evaluating architectural variations, was applied. The method is relatively easy to apply, and its results are supported by factual evidence about the candidates.

The evaluation results demonstrate that traditional scenario-based methods such as SAAM (0.27) and ALMA (0.32) score relatively low. This can be attributed to their reliance on expert judgment, susceptibility to uncertainty, the extensive involvement of non-technical stakeholders, and the lack of detail in the descriptions of architectural candidates. Additionally, these methods do not explicitly support trade-off analysis across multiple QAs.

ATAM, scoring 0.35, improves upon SAAM and ALMA by introducing trade-off analysis, yet still relies heavily on qualitative assessment. Its extension, CBAM (0.41), shows better handling of uncertainty.

The methods that scored higher – PASA (0.44) and SQUASH (0.47). They shift the focus toward quantitative metrics and experimental data. These characteristics make them more applicable for comparing SA variations in practice. However, they require a long execution time; additionally, PASA demands a large team for application, while SQUASH lacks sufficient architectural detail in candidate descriptions.

DSAV-CQRSES (0.67) outperformed other SA evaluation methods in addressing the task of comparing CQRS with ES architectural variations within the DBB Software team. This finding is supported by real-world application, which can be explained by the fact that DSAV-CQRSES was specifically developed for evaluating variations of the CQRS with ES architecture. It includes mechanisms for formal architectural modeling, trade-off analysis, and partial automation. A significant factor that limited its score was the low maturity and validation of the method, as it has only recently been introduced. Without this factor, the score would have increased to 0.76.

DOI 10.15588/1607-3274-2026-1-10

The results of the experiment can be practically applied by DBB Software team when selecting an evaluation method for real-world projects. Moreover, the qualitative and formalized assessment of the methods can be reused by other companies and teams to calculate the applicability of the considered SA evaluation methods for their specific needs.

To enhance the universality of the proposed methodology in practical applications, it is advisable to expand the set of case studies using different evaluation methods, implement multiple architectural variations for each of them, analyze the corresponding metrics (such as development and maintenance speed, performance, etc.), as well as the issues encountered by developers.

CONCLUSIONS

The scientific novelty. The article proposes a methodology for comparing SA evaluation methods. The framework for classifying and comparing SA evaluation methods has been modified for application within the context of CQRS with ES architectural variations. The proposed adapted approach allows assessing the applicability of each method based on a categorized set of variation-related questions. Each method's evaluation is represented as a multidimensional vector. The approach enables a two-stage qualitative and quantitative assessment of method effectiveness. The quantitative stage computes an applicability score using the Euclidean distance from a reference (etalon) vector *ref* to the vector of each method, considering attribute weights *w*, provided by decision-making team.

The adapted framework was applied to determine which SA evaluation method is best suited for assessing CQRS with ES variations by the DBB Software team. The methods compared were SAAM (0.27), ATAM (0.35), CBAM (0.41), ALMA (0.32), SQUASH (0.47), PASA (0.44) and DSAV-CQRSES (0.67). Based on the results of the quantitative evaluation the optimal subset $SA_{opt} = \{sa_{DSAV-CQRSES}\}$, that aligns with practical experience, as DSAV-CQRSES was specifically developed to evaluate variations of the CQRS with ES architecture.

The practical significance. The adapted framework provides a reproducible, measurable basis for selecting an SA evaluation method under real project constraints and can be instantiated for other teams and contexts by re-specifying *w* and *ref*. This, in turn, supports the selection of more appropriate architectural solutions, with positive impact on development and maintenance..

ACKNOWLEDGEMENTS

We sincerely appreciate DBB Software company [9] for providing their proprietary platform, which served as the foundation for our experiment. This platform offered essential capabilities for our research, ensuring the accuracy and reliability of our experimental results.

We also want to express our deep appreciation to Volodymyr Khandetskyi, Head of Electronic Computing Machinery department, for his valuable comments and suggestions.

DECLARATIONS

Conflict of interest: The authors declare that they have no conflict of interest in relation to this research, whether financial, personal, authorship, or otherwise, that could affect the research and its results presented in this paper.

Authors' contributions: Dmytro Hruzin: the methodology for comparing SA evaluation methods; Oleksandr Lytvynov: reviewed and discussed the proposed method and provided recommendations for improving the methodology and the manuscript's presentation.

Data availability: The manuscript has no associated data.

Software availability: The manuscript has no associated software.

Use of artificial intelligence tools: Artificial intelligence tools were used to search for and correct grammatical and punctuation mistakes in the manuscript and to improve the English translation, in parallel with other tools such as Reverso Context.

REFERENCES

1. Chrissis M. B., Konrad M., Shrum S. CMMI for Development: Guidelines for Process Integration and Product Improvement (SEI Series in Software Engineering) 3rd Edition. Boston, Massachusetts, Addison-Wesley Professional, 2011, 688 p.
2. Lytvynov O. A., Hruzin D. L.Critical causal events in systems based on CQRS with Event Sourcing architecture, *Radio Electronics Computer Science Control,* 2024, Issue 3, pp. 119–143. DOI: 10.15588/1607-3274-2024-3-11.
3. Sobhy D., Bahsoon R., Minku L. et al. Evaluation of Software Architectures under Uncertainty: A Systematic Literature Review, *ACM Transactions on Software Engineering and Methodology (TOSEM),* 2021, Vol. 30, Issue 4, pp. 1–50. DOI: 10.1145/3464305.
4. Lytvynov O. A., Hruzin D. L. Decision-making on Command Query Responsibility Segregation with Event Sourcing architectural variations, *Technology audit and production reserves,* 2025, Vol. 4, Issue 2(84), pp. 37–59. DOI: 10.15587/2706-5448.2025.337168.
5. Babar M. A., Zhu L., Jeffery R. A framework for classifying and comparing software architecture evaluation methods, *Software Engineering, Australian Conference, Melbourne, Victoria, 13–16 April 2004, proceedings*. Melbourne, Victoria, 2004, P. 309. DOI: 10.1109/ASWEC.2004.1290484.
6. Babar M. A., Gorton I. Comparison of Scenario-Based Software Architecture Evaluation Methods, *Software Engineering, 11th Asia-Pacific Conference, Busan, 30 November – 3 December 2004, proceedings*. Busan, 2004, pp. 600–607. DOI: 10.1109/APSEC.2004.38.
7. Abrahão S., Insfran E. Evaluating Software Architecture Evaluation Methods: An Internal Replication, *Conference on Evaluation and Assessment in Software Engineering (EASE 2017) : 21th International Conference, Karlskrona, 15–16 June 2017 : proceedings*. Karlskrona, 2017, pp. 144–153. DOI: 10.1145/3084226.3084253.
8. González H. J., Pelozo I., Gonzales A. et al. Validating a model-driven software architecture evaluation and improvement method: A family of experiments, Information and Software Technology, 2015, Issue 57, pp. 405–429. DOI: 10.1016/j.infsof.2014.05.018.
9. DBB Software's official company site [Electronic resource]. Access mode: https://dbbsoftware.com/.

10. Fatima I., Lago P. A Review of Software Architecture Evaluation Methods for Sustainability Assessment, *Software Architecture Companion: 20th International Conference (ICSA-C), L'Aquila, 13–17 March 2023 : proceedings*. Los Alamitos, CA: IEEE Computer Society, 2006, pp. 191–194 DOI: 10.1109/ICSA-C57050.2023.00050.
11. Sahlabadi M., Muniyandi R. C., Shukur Z. et al. Lightweight Software Architecture Evaluation for Industry: A Comprehensive Review, *Sensors*, 2022, Vol. 22, Issue 3. DOI: 10.3390/s22031252.
12. Kazman R., Klein M., Clements P. ATAM: Method for Architecture Evaluation : technical report : CMU/SEI-2000-TR-004, ESC-TR-2000-004, Product Line Systems. Pittsburgh, 2000, 71 p.
13. Kazman R., Bass L., Abowd G. et al. SAAM: a method for analyzing the properties of software architectures, *Software Engineering : 16th international conference, Sorrento, 16–21 May 1994 : proceedings.* Washington, DC, IEEE Computer Society Press, 1994, pp. 81–90. DOI: 10.1109/ICSE.1994.296768.
14. Babar M. A., Kitchenham B. Assessment of a Framework for Comparing Software Architecture Analysis Methods, *Evaluation and Assessment in Software Engineering (EASE) : 11th International Conference, UK, 2–3 April 2007 : proceedings*. Swindon, BCS Learning & Development Ltd., 2007, pp. 12–20. DOI: 10.14236/ewic/EASE2007.2.
15. Jayaratna N. Understanding and Evaluating Methodologies: NIMSAD, a Systematic Framewor. New York, McGraw-Hill, Inc., 1994, 259 p.
16. Gonzalez-Huerta J., Insfran E., Abrahão S. Models in Software Architecture Derivation and Evaluation: Challenges and Opportunities, *Model-Driven Engineering and Software Development : the 2nd International Conference, Lisbon, 7–9 January 2014 : proceedings*. Setubal, SCITEPRESS, 2014. – P. 12–31. DOI: 10.1007/978-3-319-25156-1_2.
17. Lloyd P. T. L., Galambos G. M. Technical reference architectures, *IBM Systems Journal,* 1999, Vol. 38, Issue 1, pp. 51–75. DOI: 10.1147/sj.381.005.
18. Bass L., Clements P., Kazman R. Software Architecture in Practice, Second Edition. United States, Addison-Wesley Longman Publishing Co., Inc., 2003, 528 p.
19. Lakhdissi M., Bounabat B. A New Content Framework and Metamodel for Enterprise Architecture and is Strategic Planning, *International Journal of Computer Science Issues,* 2014, Vol. 9, Issue 2. DOI: 10.1201/b16417-9.
20. Clements P., Bachmann F., Bass L. Documenting Software Architectures: Views and Beyond, Second Edition. Boston, Addison-Wesley, 2010, 592 p. ISBN: 978-0-321-55268-6.
21. Solms F. What is Software Architecture, *South African Institute for Computer Scientists and Information Technologists Conference (SAICSIT '12), Pretoria, 1–3 October 2012 : proceedings*. New York, Association for Computing Machinery, 2012, pp. 363–373. DOI: 10.1145/2389836.2389879.
22. Franchitti J. C. Enterprise Architecture Frameworks (EAFs) & Pattern Driven EAFs [Electronic resource]. Access mode: https://cs.nyu.edu/~jcf/classes/g22.3033-007/slides/session2/g22_3033_011_c23.pdf.
23. Galster M., Avgeriou P., Weyns D. et al.Variability in software architecture: Current practice and challenges, *ACM SIGSOFT Software Engineering Notes,* 2011, Vol. 36, Issue 5, pp. 30–32. DOI: 10.1145/2020976.2020978.
24. Ford N., Parsons R., Sadalage P. et al. Building Evolutionary Architectures: Automated Software Governance 2nd Edition. US, O'Reilly Media, 2022, 262 p. ISBN : 978-1492097549.
25. Dakhli S.B.D. Architectural Deviations and Inconsistencies Management: A Framework Based on Information Systems Urbanization, *Procedia Computer Science,* 2021, Vol. 181, pp. 1122–1130. DOI: 10.1016/j.procs.2021.01.309.
26. Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE). Quality model overview and usage : ISO/IEC 25002:2024. [Effective from 2024-03]. – ISO, 2024, 17 p.
27. Rumpe B. Modeling with UML: Language, Concepts, Methods. Switzerland, Springer Cham, 2016, 281 p. ISBN: 978-3-319-33933-7.
28. Chattopadhyay A., Wang Z., Martin G. E. Architecture Description Languages, Handbook of Computer Architecture. New York, Springer, 2024, Processor Design and Programming Flows, pp. 807–839. DOI: 10.1007/978-981-97-9314-3_18.
29. Lung C. H., Bot S., Kalaichelvan K. et al. An approach to software architecture analysis for evolution and reusability, *1997 conference of the Centre for Advanced Studies on Collaborative Research, Toronto, 10–13 November 1997 : proceedings.* US, IBM Press, 1997, P. 15. DOI: 10.1145/782010.782025.
30. Lassing N., Rijsenbrij D., Vliet H. On Software Architecture Analysis of Flexibility, Complexity of Changes: Size isn't Everything, *2nd Nordic Software Architecture Workshop, Ronneby, 12–13 August 1999, proceedings*. Ronneby, 1999, pp. 1103–1581.
31. Kazman R., Asundi J., Klein M.Quantifying the costs and benefits of architectural decisions, Software Engineering : the 23rd International Conference, Toronto Ontario, 12–19 May 2001, proceedings. NW Washington, *IEEE Computer Society,* 2001, pp. 297–306. DOI: 10.1109/ICSE.2001.919103.
32. Moore M., Kaman R., Klein M. Quantifying the value of architecture design decisions: Lessons from the field, *Software Engineering : 25th International Conference (ICSE03), Portland, 3–10 May 2003 : proceedings*. NW Washington, IEEE Computer Society, 2003, pp. 557–562. DOI: 10.1109/ICSE.2003.1201237.
33. Bergtsson P.O., Lassing N., Bosch J. Architecture-Level Modifiability Analysis (ALMA), *Journal of Systems and Software,* 2004, Vol. 69, pp. 129–147. DOI: 10.1016/S0164-1212(03)00080-3.
34. Ionita M. T., America P., Hammer D. K. et al. A Scenario-Driven Approach for Value, Risk, and Cost Analysis in System Architecting for Innovation, *Software Architecture : 4th Working IEEE / IFIP Conference (WICSA 2004), Oslo, 12–15 June 2004, proceedings*, P. 277. DOI: 10.1109/WICSA.2004.1310709.
35. Williams L. G., Smith C. U. PASA^{SM}: An Architectural Approach to Fixing Software Performance Problems, *28th International Computer Measurement Group Conference, Reno, 8–13 December 2002 : proceedings*, pp. 307–320.
36. Smith C. U., Williams L. G. Software performance antipatterns, *Software and performance : the 2nd international workshop (WOSP00), Ottawa, 1 September 2000 : proceedings.* New York, Association for Computing Machinery, pp. 127–136. DOI: 10.1145/350391.350420
37. Smith C. U., Williams L. G. Performance solutions: a practical guide to creating responsive, scalable software, Redwood City, CA, Addison Wesley Longman Publishing Co., Inc., 2002, 544 p. ISBN: 978-0-201-72229-1.

Received 10.09.2025.
Accepted 08.01.2026.
Published 27.03.2026.

УДК 614.2+574/578+004.38

ПОРІВНЯННЯ МЕТОДІВ ОЦІНЮВАННЯ АРХІТЕКТУРИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ В КОНТЕКСТІ АРХІТЕКТУРНИХ ВАРІАЦІЙ CQRS З EVENT SOURCING

Грузін Д. Л. – аспірант кафедри електронних обчислювальних машин Дніпровського національного університету імені Олеся Гончара, Дніпро, Україна. ROR: https://ror.org/00qk1f078. ORCID: 0009-0004-8534-2559.

Литвинов О. А. – канд. техн. наук, доцент кафедри електронних обчислювальних машин Дніпровського національного-університету імені Олеся Гончара, Дніпро, Україна. ROR: https://ror.org/00qk1f078. ORCID: 0000-0001-7660-1353.

АНОТАЦІЯ

Актуальність. Дослідження проводиться в контексті створення та обґрунтування методології оцінювання архітектури програмного забезпечення в рамках варіацій Command Query Responsibility Segregation (CQRS) з Event Sourcing (ES) архітектури.

Метою дослідження є оцінка та порівняння різних методів оцінювання архітектури програмного забезпечення з метою підтримки вибору оптимальної архітектурної варіації CQRS із ES для реальних проєктів.

Метод. Для підвищення об’єктивності процесу ухвалення архітектурних рішень застосовуються різні методи оцінювання архітектури програмного забезпечення. Проте ці методи не є універсальними, оскільки відрізняються глибиною аналізу, спрямованістю та обсягом необхідних ресурсів. Завдання, що розглядається у цьому дослідженні, полягає у виборі між архітектурними варіаціями CQRS з ES, які часто характеризуються високим ступенем структурної подібності та, відповідно, є складними для розрізнення за допомогою загальнопризначених методів оцінювання. Порівняння архітектурних варіацій потребує поглибленого аналізу, однак для більшості методів його проведення на практиці ускладнюється через часові та ресурсні обмеження. Запропонований підхід орієнтований на визначення найбільш доцільного методу оцінювання архітектури ПЗ для підтримки процесу прийняття рішень щодо вибору між варіаціями CQRS з ES. Він ґрунтується на існуючому фреймворку класифікації та порівняння методів оцінювання архітектури. Окрім якісного аналізу, підхід включає кількісну оцінку ступеня придатності до конкретного проєкту, що забезпечує більш обґрунтоване та раціональне прийняття архітектурних рішень.

Результати. Запропонований підхід було застосовано для порівняння кількох методів оцінювання архітектури ПЗ, зокрема Information Technology for Decision-making Support regarding CQRS with ES Architectural Variations (DSAV-CQRSES) – методу, спеціально розробленого для оцінки варіацій архітектури CQRS з ES.

Висновки. Існуюча система порівняння архітектур програмного забезпечення не може бути безпосередньо застосована до архітектурних варіацій (відхилень архітектури, значущих для замовника). Запропонована модифікація фреймворку орієнтовані насамперед на оцінювання варіацій архітектури CQRS з ES.

КЛЮЧОВІ СЛОВА: архітектура програмного забезпечення, порівняння методів оцінювання, CQRS з Event Sourcing, архітектурні варіації.

ЛІТЕРАТУРА

1. Chrissis M. B. CMMI for Development: Guidelines for Process Integration and Product Improvement (SEI Series in Software Engineering) 3rd Edition / M. B. Chrissis, M. Konrad, S. Shrum. – Boston, Massachusetts : Addison-Wesley Professional, 2011. – 688 p.
2. Lytvynov O. A. Critical causal events in systems based on CQRS with Event Sourcing architecture / O. A. Lytvynov, D. L. Hruzin // Radio Electronics Computer Science Control. – 2024. – Issue 3. – P. 119–143. DOI: 10.15588/1607-3274-2024-3-11.
3. Evaluation of Software Architectures under Uncertainty: A Systematic Literature Review / [D. Sobhy, R. Bahsoon, L. Minku et al.]. // ACM Transactions on Software Engineering and Methodology (TOSEM). – 2021. – Vol. 30, Issue 4. – P. 1–50. DOI: 10.1145/3464305.
4. Lytvynov O. A. Decision-making on Command Query Responsibility Segregation with Event Sourcing architectural variations / O. A. Lytvynov, D. L. Hruzin // Technology audit and production reserves. – 2025. – Vol. 4, Issue 2(84). – P. 37–59. DOI: 10.15587/2706-5448.2025.337168.
5. Babar M. A. A framework for classifying and comparing software architecture evaluation methods / M. A. Babar, L. Zhu, R. Jeffery // Software Engineering : Australian Conference, Melbourne, Victoria, 13–16 April 2004 : proceedings. – Melbourne, Victoria: 2004. – P. 309. DOI: 10.1109/ASWEC.2004.1290484.
6. Babar M.A. Comparison of Scenario-Based Software Architecture Evaluation Methods / M. A. Babar, I. Gorton // Software Engineering : 11th Asia-Pacific Conference, Busan, 30 November – 3 December 2004 : proceedings. – Busan: 2004. – P. 600–607. DOI: 10.1109/APSEC.2004.38.
7. Abrahão S. Evaluating Software Architecture Evaluation Methods: An Internal Replication / S. Abrahão, E. Insfran // Conference on Evaluation and Assessment in Software Engineering (EASE 2017) : 21th International Conference, Karlskrona, 15–16 June 2017 : proceedings. – Karlskrona: 2017. – P. 144–153. DOI: 10.1145/3084226.3084253.
8. Validating a model-driven software architecture evaluation and improvement method: A family of experiments / [H. J. González, I. Pelozo, A. Gonzales et al.] // Information and Software Technology. – 2015. – Issue 57. – P. 405–429. DOI: 10.1016/j.infsof.2014.05.018.
9. DBB Software’s official company site [Electronic resource]. – Access mode: https://dbbsoftware.com/.
10. Fatima I. A Review of Software Architecture Evaluation Methods for Sustainability Assessment / I. Fatima, P. Lago // Software Architecture Companion: 20th International Conference (ICSA-C), L’Aquila, 13–17 March 2023 : proceedings. – Los Alamitos, CA: IEEE Computer Society, 2006. – P. 191–194 DOI: 10.1109/ICSA-C57050.2023.00050.
11. Lightweight Software Architecture Evaluation for Industry: A Comprehensive Review / [M. Sahlabadi, R. C. Muniyandi, Z. Shukur et al.] // Sensors. – 2022. – Vol. 22, Issue 3. DOI: 10.3390/s22031252.
12. ATAM: Method for Architecture Evaluation : technical report : CMU/SEI-2000-TR-004, ESC-TR-2000-004 / Prod-

uct Line Systems; R. Kazman, M. Klein, P. Clements. – Pittsburgh, 2000. – 71 p.
13. SAAM: a method for analyzing the properties of software architectures / [R. Kazman, L. Bass, G. Abowd et al.] // Software Engineering : 16th international conference, Sorrento, 16–21 May 1994 : proceedings. – Washington, DC: IEEE Computer Society Press, 1994. – P. 81–90. DOI: 10.1109/ICSE.1994.296768.
14. Babar M. A. Assessment of a Framework for Comparing Software Architecture Analysis Methods / M. A. Babar, B. Kitchenham // Evaluation and Assessment in Software Engineering (EASE) : 11th International Conference, UK, 2–3 April 2007 : proceedings. – Swindon: BCS Learning & Development Ltd., 2007. – P. 12–20. DOI: 10.14236/ewic/EASE2007.2.
15. Jayaratna N. Understanding and Evaluating Methodologies: NIMSAD, a Systematic Framewor / N. Jayaratna. – New York : McGraw-Hill, Inc., 1994. – 259 p.
16. Gonzalez-Huerta J. Models in Software Architecture Derivation and Evaluation: Challenges and Opportunities / J. Gonzalez-Huerta, E. Insfran, S. Abrahão // Model-Driven Engineering and Software Development : the 2nd International Conference, Lisbon, 7–9 January 2014 : proceedings. – Setubal: SCITEPRESS, 2014. – P. 12–31. DOI: 10.1007/978-3-319-25156-1_2.
17. Lloyd P. T. L. Technical reference architectures / P. T. L. Lloyd, G. M. Galambos // IBM Systems Journal. – 1999. – Vol. 38, Issue 1. – P. 51–75. DOI: 10.1147/sj.381.005.
18. Bass L. Software Architecture in Practice, Second Edition. / L. Bass, P. Clements, R. Kazman. – United States: Addison-Wesley Longman Publishing Co., Inc., 2003. – 528 p.
19. Lakhdissi M. A New Content Framework and Metamodel for Enterprise Architecture and is Strategic Planning / M. Lakhdissi, B. Bounabat // International Journal of Computer Science Issues. – 2014. – Vol. 9, Issue 2. DOI: 10.1201/b16417-9.
20. Clements P. Documenting Software Architectures: Views and Beyond, Second Edition / P. Clements, F. Bachmann, L. Bass. – Boston : Addison-Wesley, 2010. – 592 p. ISBN: 978-0-321-55268-6.
21. Solms F. What is Software Architecture / F. Solms // South African Institute for Computer Scientists and Information Technologists Conference (SAICSIT '12), Pretoria, 1–3 October 2012 : proceedings. – New York : Association for Computing Machinery, 2012. – P. 363–373. DOI: 10.1145/2389836.2389879.
22. Franchitti J. C. Enterprise Architecture Frameworks (EAFs) & Pattern Driven EAFs [Electronic resource] / J. C. Franchitti. – Access mode: https://cs.nyu.edu/~jcf/classes/g22.3033-007/slides/session2/g22_3033_011_c23.pdf.
23. Variability in software architecture: Current practice and challenges / [M. Galster, P. Avgeriou, D. Weyns et al.] // ACM SIGSOFT Software Engineering Notes. – 2011. – Vol. 36, Issue 5. – P. 30–32. DOI: 10.1145/2020976.2020978.
24. Building Evolutionary Architectures: Automated Software Governance 2nd Edition / [N. Ford, R. Parsons, P. Sadalage et al.]. – US: O'Reilly Media, 2022. – 262 p. ISBN : 978-1492097549.
25. Dakhli S. B. D. Architectural Deviations and Inconsistencies Management: A Framework Based on Information Systems Urbanization / S. B. D. Dakhli // Procedia Computer Science. – 2021. – Vol. 181. – P. 1122–1130. DOI: 10.1016/j.procs.2021.01.309.
26. Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – Quality model overview and usage : ISO/IEC 25002:2024. – [Effective from 2024-03]. – ISO, 2024. – 17 p.
27. Rumpe B. Modeling with UML: Language, Concepts, Methods / B. Rumpe. – Switzerland : Springer Cham, 2016. – 281 p. ISBN: 978-3-319-33933-7.
28. Chattopadhyay A. Architecture Description Languages / A. Chattopadhyay, Z. Wang, G. E. Martin // Handbook of Computer Architecture. – New York : Springer, 2024. – Processor Design and Programming Flows. – P. 807–839. DOI: 10.1007/978-981-97-9314-3_18.
29. An approach to software architecture analysis for evolution and reusability / [C. H. Lung, S. Bot, K. Kalaichelvan et al.] // 1997 conference of the Centre for Advanced Studies on Collaborative Research, Toronto, 10–13 November 1997 : proceedings. – US : IBM Press, 1997. – P. 15. DOI: 10.1145/782010.782025.
30. Lassing N. On Software Architecture Analysis of Flexibility, Complexity of Changes: Size isn't Everything / [N. Lassing, D. Rijsenbrij, H. Vliet] // 2nd Nordic Software Architecture Workshop, Ronneby, 12–13 August 1999 : proceedings. – Ronneby, 1999. – P. 1103–1581.
31. Quantifying the costs and benefits of architectural decisions / [R. Kazman, J. Asundi, M. Klein] // Software Engineering : the 23rd International Conference, Toronto Ontario, 12–19 May 2001 : proceedings. – NW Washington: IEEE Computer Society, 2001. – P. 297–306. DOI: 10.1109/ICSE.2001.919103.
32. Moore M. Quantifying the value of architecture design decisions: Lessons from the field / M. Moore, R. Kaman, M. Klein // Software Engineering : 25th International Conference (ICSE03), Portland, 3–10 May 2003 : proceedings. – NW Washington: IEEE Computer Society, 2003. – P. 557–562. DOI: 10.1109/ICSE.2003.1201237.
33. Bergtsson P. O. Architecture-Level Modifiability Analysis (ALMA) / P. O. Bergtsson, N. Lassing, J. Bosch // Journal of Systems and Software. – 2004. – Vol. 69. – P. 129–147. DOI: 10.1016/S0164-1212(03)00080-3.
34. A Scenario-Driven Approach for Value, Risk, and Cost Analysis in System Architecting for Innovation. / [M. T. Ionita, P. America, D. K. Hammer et al.] // Software Architecture : 4th Working IEEE / IFIP Conference (WICSA 2004), Oslo, 12–15 June 2004 : proceedings. – P. 277. DOI: 10.1109/WICSA.2004.1310709.
35. Williams L.G. PASA^{SM}: An Architectural Approach to Fixing Software Performance Problems. / L. G. Williams, C. U. Smith // 28th International Computer Measurement Group Conference, Reno, 8–13 December 2002 : proceedings. – P. 307–320.
36. Smith C. U. Software performance antipatterns / C. U. Smith, L. G. Williams // Software and performance : the 2nd international workshop (WOSP00), Ottawa, 1 September 2000 : proceedings. – New York : Association for Computing Machinery. – P. 127–136. DOI: 10.1145/350391.350420
37. Smith C.U. Performance solutions: a practical guide to creating responsive, scalable software / C. U. Smith, L. G. Williams. – Redwood City, CA: Addison Wesley Longman Publishing Co., Inc., 2002. – 544 p. ISBN: 978-0-201-72229-1.