This study compares methods for evaluating software architecture. The work is aimed at improving the objectivity of such a comparison by using quantitative indicators obtained on the basis of a parametric model whose adequacy has been verified experimentally.

This paper proposes a methodology to quantitatively compare methods for evaluating software architecture. This methodology enabled a comparison between the architectural trade-off analysis method (ATAM) and decision support technology for architectural variations in the Command Query Responsibility Segregation (CQRS) architecture with the event sourcing approach (DSAV-CQRSES).

The methodology's feature is a parametric model built on the basis of an analysis of real projects, which makes it possible to substantiate the correctness of reference variation, as well as quantitative evaluation of the methods, in particular by the duration of application and the discrepancy of results.

The proposed methodology has made it possible to quantitatively confirm the advantages of DSAV-CQRSES over ATAM for solving the task of choosing the optimal variation of CQRSES architecture.

The duration of ATAM application was 32 hours with the participation of four specialists (total costs of 88 person-hours). The DSAV-CQRSES technology was used by one specialist for 40 hours. With the average and limit parameters of the model, DSAV-CQRSES robustly determined the recommended variation at a variance coefficient of 2.8%. In contrast, ATAM demonstrated lower accuracy and provided the correct choice in three out of five cases with a variation coefficient of 33.64%.

The results of the experiment could be used in practice to select tools for comparing variations in the CQRSES architecture

Keywords: optimization, computational experiment, ATAM, DSAV-CQRSES, data analysis, comparative analysis

UDC 004.415.2+004.412.3+004.416
DOI: 10.15587/1729-4061.2026.360407

DEVISING A METHODOLOGY TO COMPARE METHODS FOR SELECTING THE OPTIMAL ARCHITECTURAL VARIATION OF COMMAND QUERY RESPONSIBILITY SEGREGATION WITH EVENT SOURCING

Dmytro Hruzin
Corresponding author
PhD Student*
ORCID: https://orcid.org/0009-0004-8534-2559
Oleksandr Lytvynov
Candidate of Technical Sciences, Associate Professor*
E-mail: lytvynov_o@365.dnu.edu.ua
ORCID: https://orcid.org/0000-0001-7660-1353
*Department of Electronic Computing Machinery
Oles Honchar Dnipro National University
Nauky ave., 72, Dnipro, Ukraine, 49000

Received 17.02.2026
Received in revised form 28.04.2026
Accepted date 07.05.2026
Published date 30.06.2026

How to Cite: *Hruzin, D., Lytvynov, O. (2026). Devising a methodology to compare methods for selecting the optimal architectural variation of command query responsibility segregation with event sourcing. Eastern-European Journal of Enterprise Technologies, 3 (2 (141)), 24–44. https://doi.org/10.15587/1729-4061.2026.360407*

1. Introduction

As the complexity of software systems increases, the costs of their development and maintenance rise accordingly. Various architectural patterns are used to address the complexity. Among the leading solutions stands out the combination of Command Query Responsibility Segregation (CQRS) and Event Sourcing (ES) [1, 2].

Current trends [3] show that the dynamics of system development require not only flexibility and adaptability to technologies but also the capability to expand architectural solutions or even migrate to another architecture. According to [4], the architecture must be ready to respond to changing requirements, as well as to feedback from developers and end users. Such a response should take the form of controlled, gradual changes in several dimensions (technical, data, security, etc.). Changes should contribute to the evolution of the architecture in the direction of more effective ones and, at the same time, not violate any of the important architectural aspects.

The CQRS with ES architecture described in [2] has drawbacks. Attempts to eliminate these shortcomings in the context of real projects lead to the emergence of various variations of the architecture. On the one hand, these variations can be considered as modifications within the same concept. On the other hand, as independent architectures that originate from a more general one and retain the majority of its principles but differ in structural accents and a set of applied templates and methods. It is important to note that a modified architectural solution is an architectural variation only if it significantly affects the quantitative and qualitative indicators of the system. For example, it provides a significant reduction in the cost of its creation and maintenance, a noticeable improvement in its qualitative indicators (flexibility, productivity), etc.

According to [5], to optimize costs, meet quality requirements, and minimize risks associated with the development and maintenance of the system, one of the key planning stages is a well-founded choice of an architectural solution.

In the literature [6–9] tackling the issues above, methods for evaluating and comparing software architectures are

widely reported. The most widely used among them is the Architecture Tradeoff Analysis Method (ATAM) [10], which is explained by its flexibility and openness. However, the application of such methods for selecting architectural variations is limited. First, a significant part of such methods is based on expert assessments and qualitative generalizations. Second, architectural variations usually have similar structural characteristics and differ only in individual mechanisms or design solutions. Third, the same variation may be optimal for a certain class of systems and unsuitable for others. Under such conditions, even minor errors in expert assessment can significantly affect the results of ranking alternatives, that increases the probability of choosing a suboptimal architectural variation.

Thus, the existence of an objective procedure for selecting a variation based on input requirements and constraints is a relevant task in the field of development of modern complex information systems. In [11], a technology for determining optimal variations of CQRSES architecture (hereinafter, DSAV-CQRSES, Decision-making on CQRS with ES architectural variations) is described.

Taking into account the prevalence of methods for evaluating and comparing software architectures, the question arises of the possibility and effectiveness of applying these methods to the task of selecting optimal variations of the CQRSES architecture and comparing them with DSAV-CQRSES.

2. Literature review and problem statement

Among the reviewed literature sources, no papers were found that directly deal with the comparison of methods for determining the optimal architectural variation. Available publications (except for [12, 13]), firstly, consider the task of comparing methods for selecting the optimal architecture. Secondly, those papers [6–9] are aimed at classifying methods by identifying their common features and differences to facilitate further selection of the method. At the same time, the task of reasonably determining a more accurate or more convenient method for a given type of project is not set. Thirdly, the comparison in the cited studies is mainly qualitative in nature and does not cover the details of practical application and criteria for choosing a specific methodology within the chosen direction.

Thus, the framework proposed in [7] focuses on identifying similarities and differences between scenario-oriented methods for evaluating software architecture. The key principle of scenario-oriented methods is to define scenarios for quality attributes and assess risks for each scenario separately [14]. The input data are information about the evaluation methods obtained from open sources, without conducting experiments. The framework specifies a set of classification parameters in the form of questions that cover both general and specific aspects. The general ones, for example, include the number of quality attributes (QAs) considered and the estimated person-days required to apply the method. The specific ones include questions about whether the method provides support or guidance on non-technical aspects (social, organizational, etc.) that arise in the evaluation process. It should be noted that the feasibility of the classification parameters was verified in [15] by surveying experts who provide qualitative assessment. The result of applying the framework is a comparative table that contains concise information about each method considered for each classification parameter, which can be used for quick familiarization with the methods and their comparison. Further expansion of the framework is aimed at improving the survey system (classification of questions) and introducing new criteria [16].

In [8], a taxonomy of software architecture assessment methods is proposed, which is based on the conceptual classification scheme described in [7, 16]. The authors distinguish methods according to two main parameters: the artifacts to which the methods are applied, and the phases of the software life cycle during which the assessment is performed. According to the artifacts, the methods are divided into those that assess the architecture of the system as a whole and those that focus on individual architectural styles or design patterns. According to the phases, the early design phase and the late development phase are distinguished.

Paper [9] gives an overview of software architecture assessment methods with a focus on quality attributes. The basic criteria are the category of the method (scenario-oriented, experimental-oriented, etc.), the supported quality attributes, and information about practical application.

Similarly, by analyzing publications over 1990–2020, the authors of [6] offer a comparison with an emphasis on the problem of uncertainty (lack of knowledge/certainty about the factors and parameters that affect the assessment of the architecture). They offer a consistent terminology and classification scheme of approaches in terms of uncertainty accounting.

A common drawback of the considered approaches is the lack of quantitative comparison. Instead of a choice based on quantitative assessments, the decision to choose a method is largely based on the subjective interpretation of the results of a qualitative assessment by an expert. Also, the considered approaches do not provide a reasoned choice of the optimal method for a particular project. The reason for this is that such works are aimed at forming a generalized basis for choosing an assessment direction. The details of the application and criteria for choosing a specific methodology within the chosen direction are usually not considered. This makes the considered studies inappropriate for a well-founded and objective confirmation of the superiority of one method over others for solving a specific problem. An option to overcome this limitation may be to conduct an experiment with multiple application of several methods to one test task, quantitative evaluation of the results and their comparison.

This is the approach used in papers [12, 13]. The aim of this approach is to confirm the advantages of the developed Quality-Driven Architecture Derivation and Improvement (QuaDAI) method over ATAM when solving the task of building an architecture within the existing software product line (Software Product Line, SPL). This method is based on a series of experiments using ATAM and QuaDAI. Almost a hundred participants, mainly students and interns from several higher education institutions, took part in the experiments. The input data for applying the methods is a set of architectural solutions, for each of which the value of its impact on individual requirements of test projects was previously estimated. Participants used both methods to select the optimal architectural solution for the test task and provided feedback on the ease of use. The approaches were compared on six parameters.

Two parameters were measured: the average time spent on the method and the effectiveness. Effectiveness reflects how accurately a participant with relatively low qualifications can choose the appropriate architectural solution using

a certain evaluation method. It was calculated as the Euclidean distance between the n-dimensional vector of non-functional requirements values for the architecture chosen by the participant and the optimal vector of possible values. Unfortunately, the authors did not provide details on the specific values of the vectors and the method of obtaining them, as well as evidence that the reference architecture is the correct choice.

The third parameter – efficiency – was calculated as the ratio of effectiveness to the time of application. The remaining three parameters were defined according to the Technology Acceptance Model: Perceived Ease of Use, Perceived Usefulness, and Intention to Use. They were evaluated based on the results of the experiment using a Likert scale questionnaire with a set of closed questions for each variable.

Although this method is closer to our task than those considered earlier, it cannot be directly applied to solve it. The task of the QuaDAI method is not to choose between several similar architectural solutions but to build an architectural solution that satisfies the product requirements within the transformations allowed by SPL. Within the framework of the experiment, the use of QuaDAI does not involve the calculation of quantitative metrics for architectural solutions. Such metrics are given as input data. In the case of reproducing a similar experiment for DSAV-CQRSES, objective difficulties will arise due to its scale, since such an experiment requires the involvement of a significant number of participants. The involvement of participants without proper qualifications and a limited sample of projects may lead to doubts about the objectivity of the results. Changing the composition of participants or the set of non-functional requirements may significantly affect the results of the experiment.

Other shortcomings in [12, 13] include the lack of validation of reference architectural solutions. Such solutions are given a priori as input parameters of the experiment. In addition, the experiments do not provide for an explicit comparison according to the criterion of epistemic uncertainty (uncertainty of the result due to a deficit of knowledge at the stage of the modeling process), which is important for the analysis of architecture evaluation methods [6].

Given the high complexity of the full application of the methods, it is advisable to limit the research experiment to a comparison of ATAM with DSAV-CQRSES. DSAV-CQRSES is a scenario-oriented method that evaluates the architecture of the system as a whole (rather than individual architectural templates) and supports the simultaneous consideration of several quality attributes and the analysis of trade-offs between them. Therefore, it is advisable to compare it with a method based on similar principles. In this context, based on the results from [6, 8, 9, 16], ATAM was chosen as the most representative competitor. The principal argument in favor of choosing ATAM is the flexibility and openness of the methodology, which makes it possible to apply it to a wide range of tasks and is confirmed by the widespread use of ATAM [6, 12].

All this allows us to state that it is advisable to conduct a study aimed at devising a methodology for comparing software architecture evaluation methods and its application for comparing ATAM and DSAV-CQRSES methods. The methodology should provide for the following:

– involving a limited number of experiment participants who are qualified specialists;

– obtaining results supported by quantitative indicators;

– taking into account the context of application, namely the selection of CQRS architecture variations with an ES team formed from DBBSoftware employees;

– substantiating the correctness of the selection of a reference architectural variation for a given context;

– comparing methods by the level of epistemic uncertainty.

3. The study materials and methods

The purpose of our study is to devise a methodology for quantitative comparison of architectural decision analysis methods aimed at increasing the objectivity of the method selection for determining the optimal variation of the CQRSES architecture. This will make it possible to optimize software development costs and reduce the risks of unforeseen architectural changes.

To achieve the goal, the following tasks were set:

– to build a parametric model of the system for which the optimal architectural variation is mCQRS (a variation based on a snapshot of the system state) [11], and to experimentally substantiate the adequacy of the parametric model as a reflection of actual projects;

– to conduct a series of experiments to assess the optimality of applying Classical CQRS and mCQRS variations to the system (which is reflected by the parametric model) using ATAM and DSAV-CQRSES technology;

– to compare ATAM and DSAV-CQRSES according to the following parameters: level of epistemic uncertainty; application time; number of stages; output artifacts; degree of automation of calculations; reproducibility of results.

4. The study materials and methods

4. 1. The object and hypothesis of the study

The object of study is to compare methods for evaluating software architecture.

The hypothesis of the study assumes the following: compared to ATAM, DSAV-CQRSES has lower epistemic uncertainty, requires less time for application, and provides higher reproducibility of results in the task of choosing between variations. This is due to the presence of a formalized comparison mechanism and the lack of reliance on expert assessment.

The following assumptions were adopted:

1) ATAM can be used to compare variations;

2) the parametric model provides a description of the system sufficient for choosing an architectural variation.

Simplifications:

1) the parametric model is not a full-fledged system; it is a model designed to reflect the parameters of a representative system;

2) to simplify the experiment, when applying DSAV-CQRSES, the use case classes are limited to five, and the problem-space-derived QA components are limited to three [17];

3) during the experiment, only two variations of the CQRS with ES architecture are considered: mCQRS and Classical CQRS.

4. 2. Methodology for comparing software architecture assessment methods

To compare ATAM and DSAV-CQRSES, the following approach was used:

1. Select and analyze projects from the DBBSoftware knowledge base; determine the class of projects for which the study is being conducted.

2. Based on the analysis of projects, select and classify use cases.

3. According to the parameters of the use case classes, build a parametric model in the form of a frame with slots that define mandatory and optional metrics.

4. Experimentally verify the adequacy of the model: implement a prototype and ensure that the performance and development/maintenance complexity indicators are consistent with the data of real DBBSoftware projects.

5. Determine the quality attributes by which architectural variations of CQRS with ES are compared.

6. Apply ATAM to the parametric model; record the application results and time costs.

7. Apply DSAV-CQRSES to the parametric model; record the results of the application and the time costs.

8. Compare ATAM and DSAV-CQRSES according to the following criteria: level of epistemic uncertainty, application time, number of stages, output artifacts, degree of automation of calculations, reproducibility of results.

4. 3. Method for categorizing use cases

In [18], use cases are grouped by complexity level (simple-medium-complex). In [19], they are categorized by types of operations on the entity, mainly within CRUD (create, read, update, delete). Given the specifics of processes in variations of the CQRSES architecture, our paper proposes a classification based on the similarity of the algorithmic structure of the process and its purpose. The proposed algorithm for categorizing use cases consists of the following steps:

1. Collect a set of use cases from real systems. For each of them, record the purpose, input conditions, sequence of actions, expected result, subsystems involved, and execution frequency.

2. Bring the description of each use case to a single template (terminology, process boundaries, level of detail of steps). This eliminates ambiguities and improves the quality of subsequent grouping.

3. Divide the set of use cases into:

– repetitive, which are implemented according to a certain template (for example, creating new entities, such as user, doctor, patient);

– unique, which are implemented once within the system or constitute an exception (a typical example is user authorization).

4. For a subset of repetitive use cases, perform clustering by similarity of the algorithmic structure of the process and the target purpose. This involves comparing:

– sequences of process steps (for example: "input data verification → data processing → data storage");

– types of actions (creation, modification, extraction, search).

The result is classes within which use cases have a common "canonical" execution scheme.

5. For each class, determine a representative use case (standard) that reflects the typical structure and load. Add features to the class for further analysis: the average number of use cases in the system, execution frequency, expected data volumes, change frequency.

The result of the classification is:

– a list of unique use cases, allocated to a separate group. Such use cases are implemented once in each system. When selecting an architectural variation, they serve to validate candidates at the stage of their preparation. If the architectural variation makes it impossible to implement any of the unique processes, this is recorded and discussed by the evaluation team, and the corresponding variation can be removed from the list of candidates. Examples: user authorization; implementation of an event routing mechanism via an event bus;

– a set of classes of repeating use cases. The description is represented in the form of a frame [20]. Table 1 gives an example of a description of a class of entity setup commands. Examples of relevant use cases in actual projects are "user setup", "doctor setup", "patient registration", etc.

Table 1

Description of the entity setup command class

Parameter (Slot)	Essence
Typical structure	input validation → command formation → command execution → new entity setup → event publishing
Number of use cases in the first version	7–10 use cases
Factor of use cases that change during the monitoring period	≈ 68%
Number of use cases added during the monitoring period	4–7 use cases
Frequency of execution of use cases	15–25% of total record operation calls
Representative use case	Set up a doctor. Involves advanced validation of input data using third-party services and forming relationships with other entities
Monitoring period	12 sprints (approximately 6 months)

The first version is considered a minimum viable product (MVP). The monitoring period is a certain amount of time after the MVP release.

5. Results of comparing the methods for software architecture evaluation

5. 1. Parametric model of a class of projects

For experimental research, projects were selected in which the initial implementation was performed using the Classical CQRS variation. During operation and further analysis of these projects, it was found that the initially selected variation did not meet the specified requirements, in particular regarding the system's ability to be modified. As a result, migration to the mCQRS variation was performed. The selection of projects from the DBB Software knowledge base was justified by the fact that they clearly demonstrate the differences between the variations and the practical need for a reasoned choice of the optimal architectural solution.

The sample includes 5 projects: 3 from the medical domain and 2 from the marketing domain. All projects are characterized by high requirements for scalability, fault tolerance, and performance. The intensity of read operations significantly exceeds the intensity of write operations. Read queries implement complex business logic (for example, sorting objects by aggregated attributes of entities associated with them).

Based on the analysis of selected projects, an algorithm for categorizing use cases based on the similarity of the algorithmic structure of the process and the intended purpose was applied. As a result, seven classes of use cases were identified (Fig. 1). These include:

– create entity setup commands;

– data entity update commands;

– data reading requests;

– processes for achieving consistency for the corresponding entities (hereinafter referred to as consistency processes);

– processes for achieving consistency for related entities;

– processes for rebuilding projections that depend on one entity (hereinafter referred to as projection rebuilding processes);

– processes for rebuilding projections that depend on several entities.

For each class, the number of use cases, aggregates, and entities was calculated, as well as the duration of MVP preparation. Additionally, the ratio of tasks for adding new use cases and changing existing ones was estimated during the 12 sprints after the MVP release. The duration of one sprint is two working weeks. Time intervals and quantitative indicators were aggregated from the historical data of the Git version control system and the archive of the GitLab task board.

In order to simplify the experiment, the model was built for five of the seven classes. Entity setup commands, data update commands, data read requests, consistency processes, and projection reconstruction processes were considered. A description of these use case classes is given in [21].

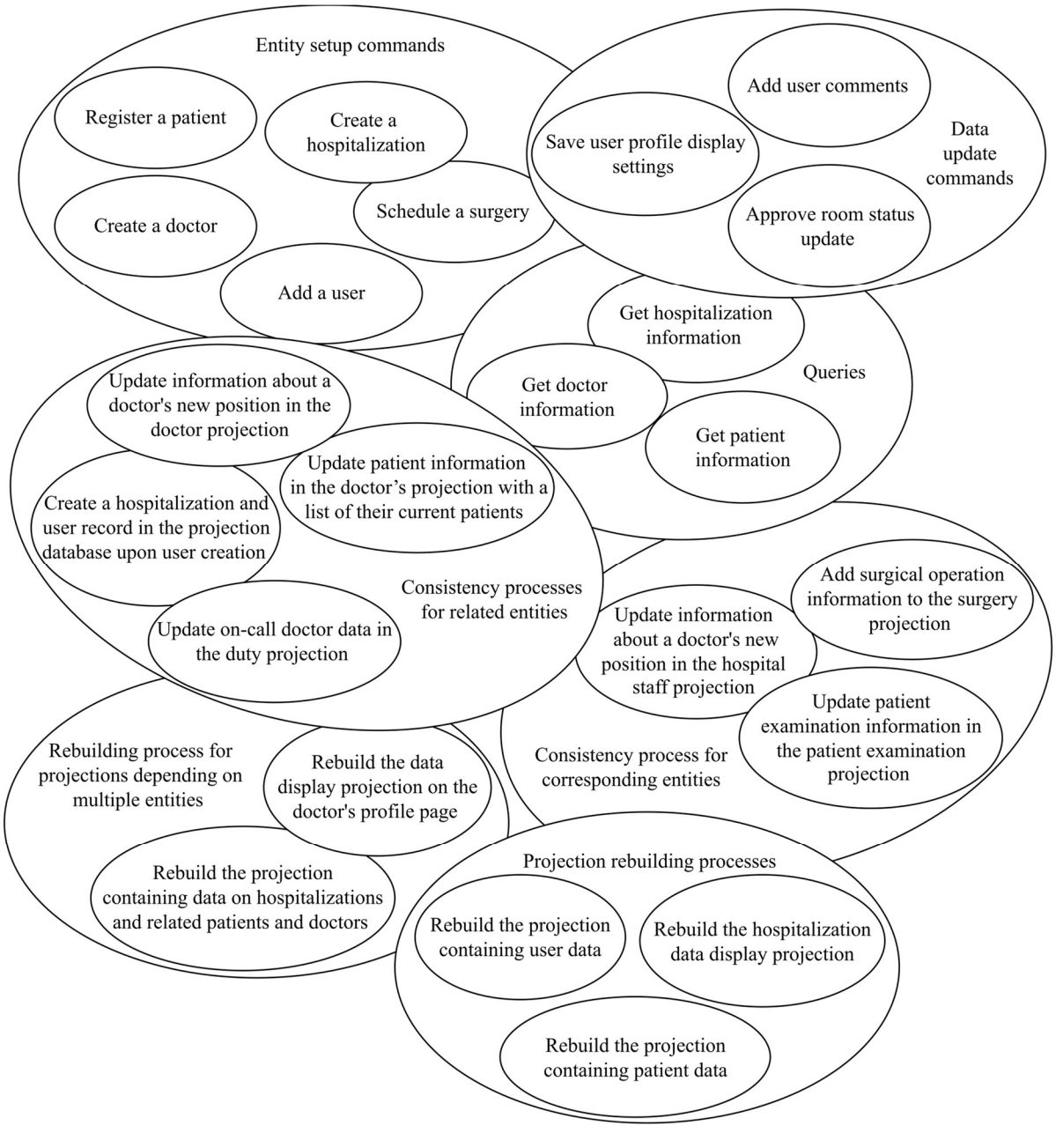

Fig. 1. Classification of use cases

According to [22], a parametric model is given by a finite set of attributes structured in the form of an ordered scheme that allows for topological ordering; variable attributes are considered as parameters. On the other hand, in [23, 24] a parametric model is defined as a parameterized class of functions $F(x; \theta)$, where θ is a finite-dimensional vector of parameters, the values of which uniquely define a specific representative of this class.

Within the framework of our study, a parametric model is used to describe a class of projects with similar parameters for which it is appropriate to use the same architectural variation. The model parameters Θ, given by the frame, form a generalized description of the project, sufficient for comparing and selecting architectural variations.

The model parameters are divided into two groups: system $\Theta_s \subset \Theta$ and context (project) $\Theta_c \subset \Theta$. System parameters characterize the structure and operating conditions of the system. Context parameters describe the conditions of development and maintenance (in particular, resources, priorities, and project constraints).

Both groups of parameters are used as input data for methods of evaluating and comparing software architectures. At the same time, system parameters are used in the parametric model function (1). This function checks whether the test system S can be considered as an implementation of a project from the class specified by the parametric model. If the values of the corresponding system parameters belong to the intervals of system parameters specified by the model $\Theta_i \in \Theta_s$, then system S corresponds to this class of projects

$$f(S) = \bigwedge_{i=1}^{n} s_i \in \left[\min(\Theta_i), \max(\Theta_i)\right], s_i \in S, \Theta_i \in \Theta_s, \quad (1)$$

where S is the set of system parameters; s_i is the value of the i-th system parameter; n is the number of system parameters of the model; Θ_s is the set of system parameters of the model; Θ_i is the interval of values of the i-th model parameter.

The proposed model is based on data on already implemented projects. The values of the model parameters are determined by the aggregated characteristics of the classes of use cases of the analyzed projects. Additionally, the metrics of these projects collected using the New Relic platform, designed for monitoring and analyzing telemetric data, were used. Individual model parameters were calculated based on the specified metrics. In particular, the ratio of read and write operations was calculated based on the data on the number of requests. Also, the share of tasks for modification during the monitoring period was calculated from formulae (2), (3):

$$t^{add} = \sum_i \overline{t_i^{add}}, \quad t^{modif} = \sum_i m_i \cdot \left(\overline{t_i^{add}} + \overline{t_i^{init}}\right), \quad (2)$$

$$p^{modif} = \frac{t^{modif}}{t^{modif} + t^{add}}, \quad (3)$$

where $\overline{t_i^{init}}$ is the average number of use cases implemented during the development of MVP; t_i^{add} – average number of tasks for adding use cases during the monitoring period; m – proportion of use cases that are modified after creation; t^{add}, t^{modif} – total number of tasks for adding and modifying use cases during the monitoring period; p^{modif} – proportion of tasks for modifying use cases from the total number of tasks during the monitoring period.

The parameters are summarized in Table 2.

Table 2

Parametric model of a class of projects for which the optimal architectural variation is mCQRS

Parameter	Measurement unit	Value
System settings		
Number of commands in the MVP stage	unit	34–45
Create class commands	unit	7–10
Update class commands	unit	27–35
Number of requests in the MVP stage	unit	16–29
Number of event handlers in the MVP stage	unit	36–52
Number of projections in the MVP stage	unit	12–22
Read request intensity	request/min	2200–2500
Write request rate	request/min	91–247
Read request rate	%	91–96
Write request rate	%	4–9
Event write rate	event/min	98–336
Average number of events per unit	event/unit	43–142
Response time for read and write requests (p95)	ms	43–66
Response time for read and write requests (p99)	ms	148–387
Record request success rate	%	> 99,909
Read request success rate	%	> 99,923
Contextual (project) parameters		
Frequency of change of event types	1/sprint	1–3
Frequency of deletion of user data in accordance with the requirements by the General Data Protection Regulation (GDPR) [25]	1/sprint	0,5–0,66
Relative business priority of ease of development and maintenance versus system performance	point (0–100)	> 50
Duration of MVP development	sprints	6–8
Support period for which evaluation is performed	sprints	12
Adding new functions (during the monitoring period)	%	~ 38
Modifying functions (during the monitoring period)	%	~ 62
Estimated labor costs for modifying one system process (changing logic or fixing errors)	person-hour	< 2
Estimated work costs for adding a new entity along with typical use cases for processing it. The work includes: 1 creation process, 3–5 update processes, 4–7 consistency processes, 2–4 read queries, and 1 projection rebuild process	person-hour	< 8

This parametric model covers the parameters necessary for conducting the experiment within the framework of this work. To solve other tasks, the model can be expanded with additional parameters. In particular, it can include parameters related to quality attributes, such as security or availability. For example, system availability at the level of 99.9% means an allowable downtime of approximately 8 hours 46 minutes per year.

To substantiate the adequacy of the formed parametric model of the class of projects for which the mCQRS variation is optimal, two approaches to validation are considered.

The first approach consists in forming a significant number of new projects and implementing them in accordance

with the specified model parameters. The disadvantages of this approach are significant costs of time and resources. In addition, increasing the number of projects does not guarantee the adequacy of the model, since there is still a probability of the appearance of a new project with characteristics that go beyond its limits and require a revision of the parameterization.

The second approach involves building a typical software system, the parameters of which are coordinated with the parameters of the model, and its functionality is formed by selecting and generalizing the characteristic functions of actual projects, grouped by classes of use cases.

Parametric model implementation plan:

1. Based on the model parameters, form a service-level agreement (SLA) [26]. Determine the operating conditions and the composition of the development team.

2. Within the second approach, design two test systems: one using the Classical CQRS variation, the other – mCQRS. Implement a set of functions that corresponds to the specified model parameters in terms of volume and structure.

3. Evaluate the parameters of the test systems. Using formula (1), confirm that these systems can be considered as implementations of a project from the class specified by the parametric model.

4. Evaluate the complexity of the implementation. Check the systems' compliance with the SLA requirements.

Definition of the SLA of a project implementing the parametric model:

– Performance SLA (according to the Apdex methodology [27]);

– satisfactory response time for write requests (p95) – 100 ms;

– allowable response time for write requests (p99) – 400 ms;

– SLA of development speed;

– modification of one system process (change of logic or correction of errors) – up to 2 person-hours;

– addition of a new entity together with typical use cases of its processing – up to 8 person-hours. The scope of work includes: 1 setup process, 3–5 update processes, 4–7 consistency processes, 2–4 read requests, and 1 projection reconstruction process;

– SLA of service availability;

– the share of successful requests is not less than 99.9% of the total number of requests.

Operating conditions:

– approximate load: 100 requests per minute for writing and 2300 requests per minute for reading;

– deployment: server with 8 GB of RAM and 2 vCPUs (virtual Central Processing Unit).

Determine the team of developers of test systems:

– team composition: 1 developer;

– role: backend developer;

– qualification level: senior developer (senior) on the scale of junior – middle – senior [28];

– commercial experience: 11 years;

– education: Master's degree in computer systems and networks;

– experience with CQRSES: 7 years.

The designed systems are available for viewing in an open repository at the link given in [29]. To confirm the quality of the system code, the SonarCloud service was used. In terms of security, reliability, and maintainability, the systems received the highest scores, the code coverage by tests is 85.5%. Code quality metrics are available at [30].

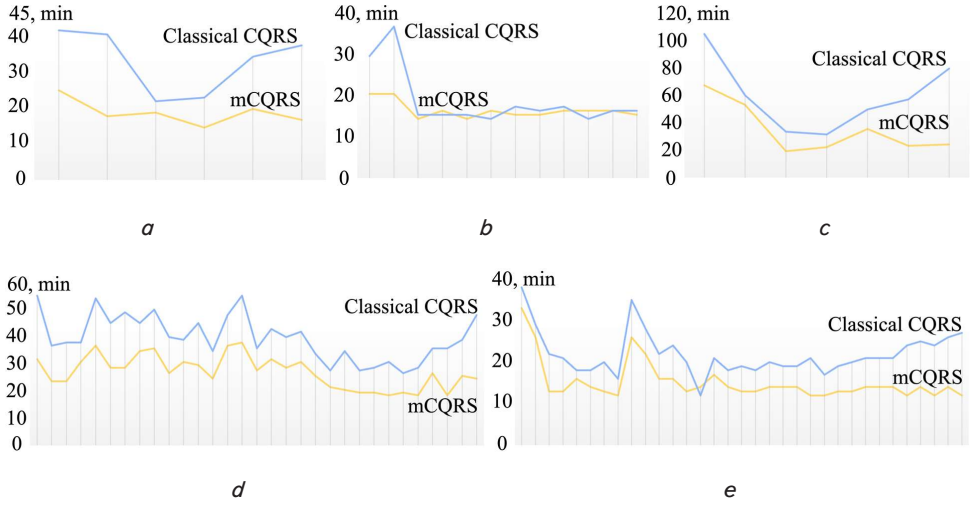

Fig. 2. Duration of development of functional components during design of systems implementing a parametric model: *a* – setup; *b* – reading; *c* – reconstruction of projection; *d* – updating; *e* – achieving consistency

During the development of systems implementing the parametric model, the time spent on the implementation of individual components was measured. Based on the results, plots of the time (in minutes) required for the implementation of each individual process were constructed (Fig. 2).

The plots show similar dynamics (except for read requests). The resulting curves can be conditionally divided into three phases:

Phase 1. The development of the first few processes of each category requires a significant amount of time, as it involves the initial implementation of the basic components. During the implementation of components for subsequent processes, the existing code is usually reused, while simultaneously refining and improving the initial implementations.

Phase 2. Then there is an interval of maximum efficiency, due to the reuse of components. At this stage, a critical mass of dependences has not yet been formed, which makes it difficult to make changes.

Phase 3. With the accumulation of components and interdependences, the development pace decreases. Due to the higher structural complexity of Classical CQRS, within this variation, a tendency towards an increase in the duration of process implementation is observed; for mCQRS, a corresponding trend is practically not observed. This is most clearly seen in the plots in Fig. 2, *a*, *c*, *d*.

Load testing was carried out for both developed systems. 100 iterations were performed for each system. In each iteration, requests were sent to each system in accordance with

the SLA requirements: 2300 read requests and 100 write requests per minute.

According to the test results, the parameter values for each system parameter of the model were determined. Both systems have the same number of commands, requests, event handlers, and projections in accordance with the parametric model implementation plan. The intensity and ratio of read and write requests were set in accordance with the SLA requirements. The intensity of event recording was defined as the ratio of the total number of events generated during testing to the number of iterations (100). The response time for the p95 and p99 percentiles was also calculated. The proportion of successful requests was 100% for both systems.

According to formula (1), the compliance of both systems with the system parameters of the model was confirmed. The results are given in Table 3. The range of model parameter values is taken from Table 2.

Table 3

Compliance of systems with a parametric model

Parameter	Measurement unit	Value range	Classical CQRS		mCQRS	
Number of commands in the MVP stage	unit	34–45	38	+	38	+
Create class commands	unit	7–10	7	+	7	+
Update class commands	unit	27–35	31	+	31	+
Number of requests in the MVP stage	unit	16–29	20	+	20	+
Number of event handlers in the MVP stage	unit	36–52	46	+	46	+
Number of projections in the MVP stage	unit	12–22	12	+	12	+
Read request rate	request/min	2200–2500	2300	+	2300	+
Write request rate	request/min	91–247	100	+	100	+
Read request rate	%	91–96	95,8	+	95,8	+
Write request rate	%	4–9	4,2	+	4,2	+
Event write rate	event/min	98–336	109	+	109	+
Average number of events per unit	event/unit	43–142	109	+	109	+
Read and write request response time (p95)	ms	43–66	58	+	65	+
Read and write request response time (p99)	ms	148–387	156	+	297	+
Write request success rate	%	> 99,909	100	+	100	+
Read request success rate	%	> 99,923	100	+	100	+

The systems were checked for compliance with SLA requirements. For this purpose, the average values of development labor parameters were calculated. The calculation was performed based on measurements of the time spent on implementing individual components of the test systems. The results are summarized in Table 4.

Table 4

Compliance of systems with SLA requirements

Metric	Phase	Classical CQRS	mCQRS	Threshold value
Productivity				
p95, ms	3	58	65	100
p99, ms	3	156	297	400
Labor costs for development				
Average duration of modification of one system process, min	2	26,6	20,2	60
	3	40,8	18,2	60
Adding a new typical use case to the system, h	2	6,82	5,45	8
	3	10	5,1	8
Successful request rate, %	3	100	100	99,9

According to the results of the SLA compliance assessment, it was found that for mCQRS all indicators meet the specified constraints. For Classical CQRS, most indicators also meet the specified constraints. At the same time, the labor costs for adding a new typical use case to the system slightly exceed the specified threshold value. Such an excess is expected, since the SLA requirements are formed on the basis of the contextual parameters of the model, which corresponds to the mCQRS variation. This additionally confirms that the test systems can be considered as implementations of a project from the class specified by the parametric model.

Thus, the experimental results indicate the adequacy of the parametric model for describing the class of actual projects. They also confirm that for projects whose parameters correspond to the specified parametric model, the mCQRS variation is more appropriate than Classical CQRS according to the considered metrics.

5. 2. Results of experiments to determine the optimal architectural variation

Key QAs for CQRS with ES are derived as follows.

First, despite the importance of the full spectrum of QA, in professional sources on CQRS with ES, the main advantages of this architecture are termed productivity and scalability [31, 32]. The key disadvantages are associated with the introduction of additional complexity into the system [33]. Accordingly, architectural variations are mainly aimed at achieving two goals: reducing complexity and increasing performance, that is, reducing the cost of development and maintenance and accelerating the system. Analysis of examples of actual projects revealed that complexity directly affected the requirements for team qualifications and the costs of design and implementation. At the same time, performance requirements often set strict limits. If the performance indicators did not meet them, the architecture had to be significantly modified or replaced with another solution.

Second, approaches like ATAM explicitly prioritize only the riskiest quality attributes using a “utility tree” and trade-off analysis instead of exhaustively evaluating every attribute listed in standard quality models [10].

Third, according to [34], quality attributes are most suitable for architectural evaluation when they have high explainability and reproducibility. In the context of CQRS with ES, where variations represent implementations of the same approach, many factors (e.g., testability, connectivity, deployability, etc.) tend to be nearly identical and are provided as mandatory infrastructure requirements. Thus, in this context, such attributes have low explainability.

Thus, in our study, the evaluation is limited to two quality attributes – performance and complexity – due to their high discriminating power between variations and their direct relationship to project costs and risks. The remaining qualities were considered as strict requirements for the system and did not significantly influence the choice.

Application of ATAM.

Five teams of three specialists each were used to apply the ATAM method. Information on the positions and experience of the participants is given in Table 5.

Table 5

Information about the positions and experience of specialists who carried out the assessment using the ATAM method

Specialist ID	Position	Experience, years
1, 4, 7, 13	Senior Developer	over 10
10, 11		8
6, 9, 12	Mid-level developer	6–7
2, 3, 5, 8, 14, 15		3–5

According to [10], the ATAM method application plan is as follows:

– presentation of the ATAM method for teams;
– definition of business drivers and goals;
– presentation of the architecture;
– identification of candidate solutions;
– construction of a "utility tree";
– analysis of candidate solutions;
– definition and prioritization of scenarios;
– analysis of candidate architectural solutions;
– presentation of results.

8 person-hours were spent on preparing presentations of the ATAM method, architecture, and candidate solutions. The presentation was held for all teams in parallel; materials are available in [35]. Additionally, business drivers and business goals were identified within the framework of a joint session. 8 hours were spent on conducting this session with the participation of 16 specialists.

The key business driver was determined to be the modifiability of the system since a prompt response to a dynamic and highly competitive market is required. Business goals included business support and process automation, in particular, recording customers, sending notifications, displaying customer information with the ability to search and view the history of interaction with the customer.

Quality attributes were analyzed separately by each team. Fig. 3, 4 show generalized descriptions of quality attributes with the separation of incentives, parameters, and responses in accordance with ATAM recommendations.

Each team separately constructed a utility tree. Fig. 5 shows the generalized utility tree.

After analyzing the candidate solutions, each team, based on expert assessment, determined the priorities for each leaf of the utility tree (Table 6) according to two criteria:

– the importance of the node for the success of the system;

– the level of risk associated with achieving a given level of indicator (that is, an assessment of the difficulty of achieving the corresponding performance or ability to modify).

The assessment was performed on a three-level scale: low – medium – high (L/M/H).

Table 6

Priorities of the leaves in the utility tree on the L/M/H scale

Leaf of the tree of utility	Team				
	1	2	3	4	5
Performing update operations under normal load	M, M	M, L	M, L	M, L	M, M
Performing update operations during peak load	M, H	L, M	L, M	L, M	L, M
Performing read operations under normal load	M, L	M, L	M, L	M, L	M, L
Performing read operations under peak load	M, H	L, L	L, M	M, H	L, H
Reconstruction of one projection	L, L	M, M	M, L	H, M	L, L
Reconstruction of all projections	L, L	M, H	M, M	M, H	L, M
Adding a typical write process	H, L	M, L	H, L	H, M	M, L
Adding a non-typical write process	L, M	M, M	L, H	L, M	L, M
Adding a typical read process	H, L	M, L	H, L	H, M	M, L
Changing business logic in an existing process without changing event contracts	M, L	H, M	H, M	H, L	M, L
Event type evolution	M, L	M, L	L, L	L, L	M, H
Data deletion according to GDPR	L, M	M, L	L, M	L, L	L, L
Adding a new aggregate	M, L	M, M	H, H	M, L	M, M

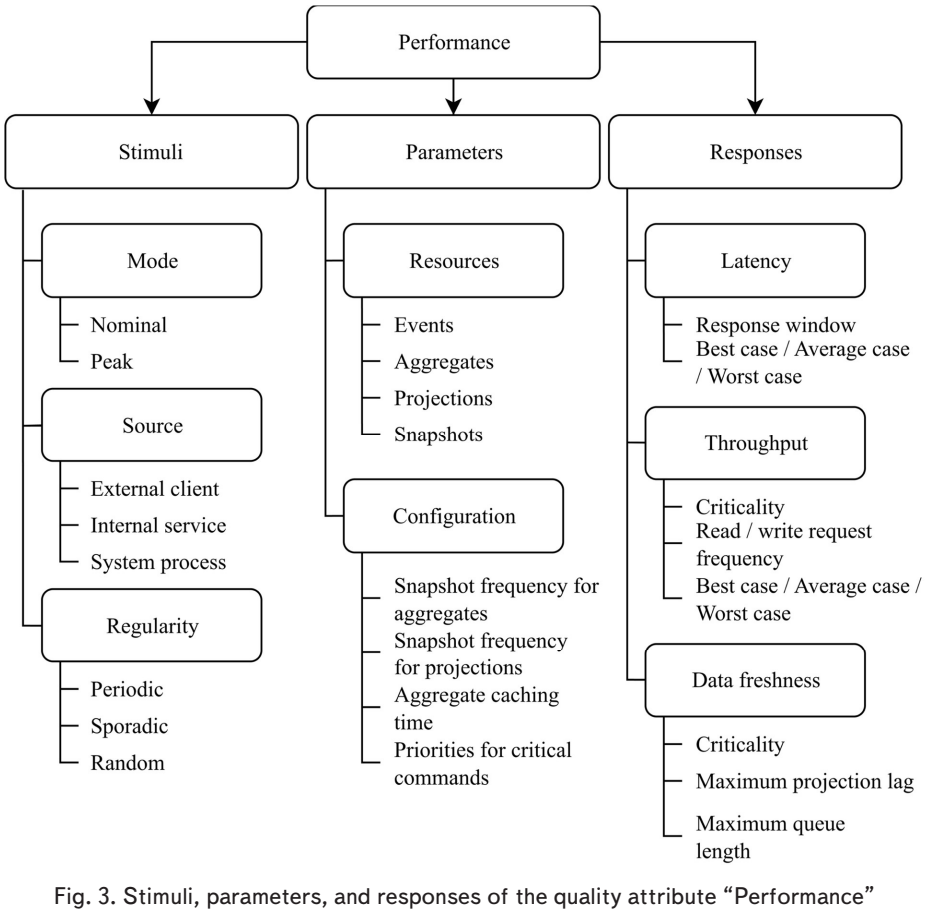

Fig. 3. Stimuli, parameters, and responses of the quality attribute "Performance"

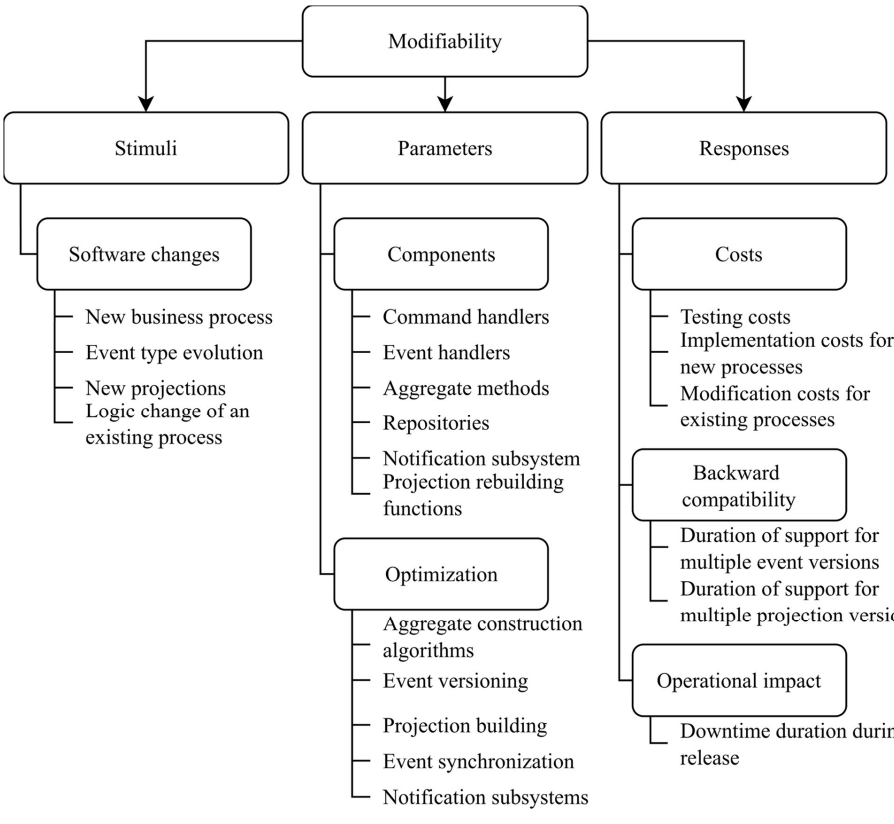

Fig. 4. Stimuli, parameters, and responses of the quality attribute "Modifiability"

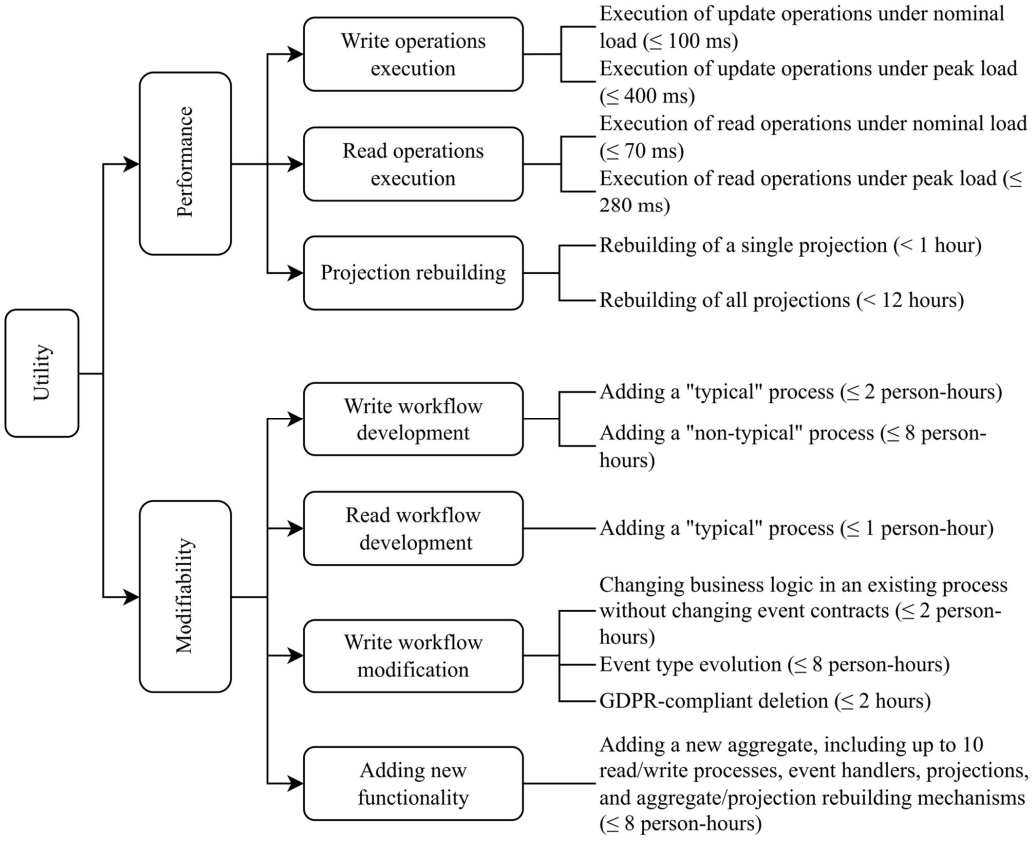

Fig. 5. A generalized utility tree built on the basis of the utility trees generated by each team

Separately, a list of scenarios was generated based on the results from Scenario Brainstorming (Table 7). All 16 specialists participated in the session, as ATAM recommends this approach for large groups.

Table 7

Scenarios identified during Scenario Brainstorming

No.	Type	Scenario description
1	Growth development scenario	Adding a typical process: regular work; team level; 1 aggregate; 1–2 projections; labor intensity ≤ 8 man-hours; ≤ 5 modules changed
2	Development scenario	Changing business logic in an existing process: regular work; aggregate method, event handler, 1 projection; labor intensity ≤ 4 man-hours; ≤ 1 release; no change to event contracts
3	Development scenario	Adding new functionality to the system up to 10 commands and queries; read/write API, aggregate(s), projections (2–3), indexes; labor intensity ≤ 20 man-hours.
4	Development scenario	Event type evolution (v → v+1): working system; event repository, handlers, migration tools; processing versions v−1...v+1; downtime < 30 min; reconciliation of all projections ≤ 24 h; no loss of events
5	Use case scenario	GDPR data deletion request execution: production system; event store, snapshots, projections; user data deleted or anonymized; completion ≤ 2 h; full convergence of projections ≤ 24 h
6	Research scenario	Migration mCQRS ↔ classic CQRS with ES within the same context: planning period; migration pipelines; parallel mode with correctness check; duration ≤ 2 sprints; window ≤ 14 days; downtime < 30 min
7	Use case scenario	Read operation execution under peak load: API, projection and cache layers; response time ≤ 280 ms
8	Use case scenario	Update operation execution under normal load: command handler, repository, event store, snapshots, event bus; acknowledgement ≤ 100 ms
9	Use case scenario	Performing an update operation at peak load: command handler, repository, event store, sna shots, event bus; acknowledgement ≤ 400 s
10	Use case scenario	Processing dependent events, no false states in critical threads; detected missynchronizations are eliminated automatically or by rebuilding projections
11	Use case scenario	Full rebuilding of projections in the produ tion system; rebuilding duration ≤ 24 hours
12	Exploratory research scenario	Data loss in case of failure during a query to the event store
13	Research scenario	10x increase in system workload. What changes does the system architecture require, other than changing or scaling the deployment infrastructure?
14	Research scenario	How does accumulating a large number of events (> 1000 for each unit) in the event store affect performance?
15	Development scenario	Development of a non-standard process: team level; 1 unit; 1–2 projections; labor intensity ≤ 40 man-hours; ≤ 5 modules changed

Each team separately voted on the scenarios. Each team member chose the five scenarios they considered most important. The results of the voting are given in Table 8.

Scenarios that were voted for by at least two experts within the team were considered priority. For each such scenario, sensitivity points, compromise points, and risks were identified. The data obtained are summarized and given in Table 9.

Table 8

Voting results on the importance of scenarios

No.	Scenario	Team				
		1	2	3	4	5
1	Adding a typical process	2	1	2	3	1
2	Changing business logic in an existing process	0	2	2	2	1
3	Adding new functionality to the system	0	2	3	0	2
4	Evolution of an event type	1	1	0	0	3
5	Deleting data according to GDPR	1	0	0	0	0
6	Migration mCQRS ↔ Classical CQRS within the same context	1	0	0	0	0
7	Performing a read operation under peak load	2	0	1	2	2
8	Performing an update operation under normal load	3	1	1	0	2
9	Performing an update operation under peak load	2	1	1	1	0
10	Processing dependent events	1	2	2	2	2
11	Complete reconstruction of projections	0	2	1	3	0
12	Data loss in case of failure during a query to the event store	1	1	0	2	0
13	Increase in the standard load on the system by 10 times	0	0	1	0	1
14	Impact of accumulation of a large number of events on system performance	0	2	0	0	0
15	Development of a non-standard process	1	0	1	0	1

Based on the considered sensitivity points, compromise points, and risks, the teams selected the optimal CQRSES architecture variation for application (Table 10).

In order to quantitatively assess the impact of negative factors (by analogy with [36]) on the decisions of the experiment participants, the weight of these factors (w) was calculated using formula (4) and is given in Table 11. A larger value of w corresponds to a larger total weight of factors acting against the choice of the corresponding variation

$$w = \sum_i \left(v_i \cdot f\left(S_i\right) \right), \tag{4}$$

where w is the weight of negative factors; S_i is the scenario; v_i is the number of votes for scenario S_i; $f(S_i)$ is the function for calculating the sum of negative factors for scenario S_i within the corresponding variation.

The discrepancy of values is estimated using the standard deviation and the coefficient of variation, calculated from formulae (5) to (7):

$$\bar{x} = \frac{1}{n} \sum_{j=1}^{n} x_j, \tag{5}$$

$$s = \sqrt{\frac{1}{n-1} \sum_{j=1}^{n} \left(x_j - \bar{x} \right)^2}, \tag{6}$$

$$Cv = \frac{s}{\bar{x}}, \tag{7}$$

where n is the number of values, s is the standard deviation, Cv is the coefficient of variation.

Table 9

Sensitivity points, compromise points, and risks

No.	Description	Type	Variation
1	Complexity of adding functionality ↔ write speed	Tradeoff point of compromise	Between variations
2	The duration of adding a typical process can exceed SLA limits, especially for Classical CQRS	Sensitivity point	Classical CQRS
3	The duration of changing business logic in an existing process can exceed SLA limits, especially for Classical CQRS	Sensitivity point	Classical CQRS
4	The duration of adding new functionality to the system can exceed SLA limits, especially for Classical CQRS	Sensitivity point	Classical CQRS
5	For mCQRS, there is a risk that when event types change, developers will not provide correct support for previous versions. In this case, reproducing events (if necessary) may be difficult or impossible	Risk	mCQRS
6	**3** and due to the lack of a well-established process for deploying a new event store in Classical CQRS, the evolution of the event type is highly complex (compared to mCQRS)	Sensitivity point	Classical CQRS
7	Both variations do not provide built-in means of prioritizing read requests during peak load. It is assumed that caching and denormalized data structures reduce the risk of loss of service availability. At the same time, it is advisable to provide mechanisms for prioritization, scaling, and queue management	Sensitivity point	Both variations
8	Update execution times can exceed SLA limits, especially for mCQRS	Risk	mCQRS
9	Update operations can take longer than SLA limits. An additional factor for mCQRS is the need to not only commit events to the event store but also create or update records in the snapshot database	Risk	mCQRS
10	Both variations do not provide built-in means of prioritizing write requests during peak load. It is advisable to provide scaling and queue management mechanisms, as well as request prioritization policies	Sensitivity point	Both variations
11	The consequences of incorrect processing of dependent events are usually eliminated by rebuilding projections; in Classical CQRS, such an operation is usually more complicated	Risk	Classical CQRS
12	Both variations do not offer a ready-made solution to the problem of processing dependent events. If such dependences exist, specialized mechanisms for their coordination should be considered	Sensitivity point	Both variations
13	A complete reconstruction of projections in Classical CQRS can take a significant amount of time (more than 24 hours)	Risk	Classical CQRS
14	For mCQRS, full recovery of the database and projections by replaying events (if necessary) is not guaranteed	Sensitivity point	mCQRS
15	In both variations, there is a risk of data loss in the event of a failure while writing to the event store or snapshot database	Risk	Both variations
16	To prevent data loss, it is advisable to use reliable delivery and storage mechanisms, such as message queues or backup stores. In mCQRS, implementing such mechanisms can be easier due to the separation of stores	Sensitivity point	Classical CQRS
17	The impact of accumulating a large number of events on performance in Classical CQRS needs further empirical confirmation. For mCQRS, this impact is expected to be less pronounced	Sensitivity point	Classical CQRS

Table 10

Distribution of sensitivity points, compromise points, and risks by team and selected variation

Team	Sensitivity points, Compromise points, Risks	Selected variation
1	1, 2, 7, 8, 9, 10	Classical CQRS
2	1, 3, 4, 11, 12, 13, 14, 17	mCQRS
3	1, 2, 3, 4, 11, 12	mCQRS
4	1, 2, 3, 7, 11, 12, 13, 14, 15, 16	mCQRS
5	1, 4, 5, 6, 7, 8, 11, 12	Classical CQRS

Table 11

Assessment of sensitivity points, tradeoff compromise points, and risks by teams and variations

Team	1	2	3	4	5	*s*	*Cv*, %
Against Classical CQRS	3	6	5	8	5	1.82	33.64
Against mCQRS	4	2	1	4	4	1.41	47.14

The duration of application of the ATAM method is given in Table 12.

Table 12

Duration of ATAM use

Activity	Number of specialists	Duration, hours	Labor intensity, person-hours
Preparation of architectural candidate solutions	1	8	8
Presentation of ATAM and candidate solutions	4	8	32
Comparison and decision making	3	16	48
Together	4	32	88

DSAV-CQRSES application.

According to [11], the process of applying DSAV-CQRSES technology to support decision-making regarding the choice of a CQRS architecture variation from an ES architecture is as follows:

– preparation of architectural variations for evaluation;
– familiarization with the technology;
– identification of quality attributes;
– evaluation of quality attributes, which includes:
– formalization of architectural variations;
– evaluation of processes;
– determination of metrics that require the development of a representative test project (RTP);
– evaluation of variations;
– cost-benefit analysis;
– comparison of variations;
– provision of recommendations.

12 hours were spent on preparing architectural variations for evaluation and familiarization with the technology.

Quality attributes (complexity and performance) were decomposed according to [17]. Based on the use case classes, the operational components of quality attributes were identified:

– complexity of developing / modifying use cases for the entity setup command class;
– complexity of developing / modifying use cases for the data update command class;
– complexity of developing / modifying use cases for the data read request class;
– complexity of developing / modifying use cases for the consistency achievement process class;
– complexity of developing / modifying use cases for the projection reconstruction process class.

The components determined by the problem space were defined taking into account the typical problems of the CQRSES architecture described in [11], namely:

– performance of read operations;
– performance of write operations;
– complexity of implementing use cases of event type evolution;
– complexity of implementing use cases of deleting user data in accordance with GDPR requirements.

To calculate the components of the quality attribute "Complexity", the processes of the evaluated architectural variations were formalized using the knowledge representation framework model [11]. The description format is consistent with the representation of the use case class structure given in [37]. The resulting formalized descriptions are given in [38].

The development and modification complexity components include two components: design complexity and implementation complexity.

The design complexity is calculated using the cognitive functional complexity method [39], which is based on a process flowchart and involves assigning a unit of cognitive weight (CWU) to each block. The complexity of the business logic of individual functions is not taken into account during the calculations since it is the same for both variations. Details of the calculations are provided in [38]; the results are given in Table 13.

Table 13

Design complexity

Activity	Classical CQRS (CWU)		mCQRS (CWU)	
	Development	Modification	Development	Modification
1	2	3	4	5
Create setup command class activities				
Command formation	2	2	2	2
Command validation	4	1	4	1
Command routing	1	1	1	1
Creating an aggregate	3	3	3	3
Saving the aggregate	6	0	10	0
Publishing events	2	0	2	0
Total	18	7	22	7
Update command class activities				
Command formation	2	2	2	2
Command validation	4	1	4	1
Command routing	1	1	1	1
Receiving the aggregate	24	8	10	4
Aggregate status update	6	6	6	6
Saving the aggregate	16	6	10	0
Publishing events	2	0	2	0
Total	55	24	35	14
Activities of the process class of achieving consistency				
Event routing	1	1	1	1
Event processing	21	4	12	4
Sending notifications	4	2	4	2
Total	27	7	17	7
1	2	3	4	5

Continuation of Table 13

Data read request class activities				
Forming a request	2	2	2	2
Request validation	4	1	4	1
Getting a projection	4	4	4	4
Converting a projection to a DTO	2	2	2	2
Total	12	9	12	9
Activities of the projection reconstruction process class				
Processing the request	1	1	1	1
Getting information about the projection	4	4	–	–
Getting a snapshot	4	0	6	2
Getting events	4	0	–	–
Building a Classical CQRS projection	24	12	4	2
Updating a projection in the database	4	0	4	0
Total	41	17	15	5
User data deletion activities in accordance with GDPR requirements				
Creating an event repository	1	–	–	–
Implementing the transformation function	2	–	–	–
Starting data migration	3	–	–	–
Anonymizing data in a snapshot	–	–	1	–
Anonymizing data in an event store	–	–	1	–
Rebuilding projections	3	–	1	–
Testing	2	–	1	–
Total	11	–	4	–
Event type version update operation activities				
Creating an event repository	1	–	–	–
Adding a new version of the event type	1	–	–	–
Implementing the transformation function	2	–	–	–
Starting data migration	3	–	–	–
Updating event type version in snapshot	–	–	2	–
Rebuilding projections	3	–	1	–
Testing	2	–	2	–
Total	12	–	5	–

In these calculations, the complexity of the activity of the projection reconstruction process for Classical CQRS is taken as the lower limit. The complexity of the reconstruction depends on the number of event types to which the projection is subscribed. The calculations assume 4–6 event types, while in actual projects for individual projections their number can reach dozens.

The assessment of the complexity of the implementation was performed based on the actual time spent on creating RTP functional components, which in our work is the implementation of the parametric model (Table 14). For each component, the implementation time in minutes was recorded, and the average implementation time was calculated based on aggregated values grouped by use case classes.

The values of the development/modification complexity components were calculated by aggregating the values of design complexity and implementation complexity according to the overall complexity calculation algorithm [11]. The results are given in Table 15.

The performance metric was calculated based on the server response time indicators (Table 16) obtained during the operation of the system under experimental conditions. The system was deployed on a server with 8 GB of RAM and 2 vCPUs. The metric was calculated using the following algorithm:

1. By analogy with [40], stakeholders determine acceptable performance parameters (classes). For each class, a score is determined that characterizes the performance level. This work uses the gradation given in Table 17.

2. Using fuzzy logic methods [41], for each variation, a performance estimate is calculated based on the average server response time metric using formulae (8), (9). A graphical representation of the calculations is shown in Fig. 6:

$$\mu_L(t)=\frac{t_R-t}{t_R-t_L},\quad \mu_R(t)=\frac{t-t_L}{t_R-t_L}, \tag{8}$$

$$p(t)=s_L\cdot\mu_L+s_R\cdot\mu_R, \tag{9}$$

where p is the performance indicator; s_L, s_R are the scores of the left and right threshold values; t is the average server response time; t_L, t_R are the left and right threshold values; μ_L, μ_R are the membership functions of the value t to the corresponding classes.

Table 14

Implementation complexity

Activity	Classical CQRS, minutes	mCQRS, minutes
Create setup command class activities		
Command formation	3.67	3
Command validation	3.5	1.83
Command routing	1.83	1.33
Creating an aggregate	13.67	8
Saving the aggregate	7	2.67
Publishing events	2.17	1.17
Total	31.83	18
Update command class activities		
Command formation	3.32	3
Command validation	3.1	3.03
Command routing	1.68	1.26
Receiving the aggregate	12.9	3.71
Aggregate status update	10.39	10.19
Saving the aggregate	5.74	4
Publishing events	1.68	1.19
Total	38.81	26.39
Activities of the process class of achieving consistency		
Event routing	1.91	1.3
Event processing	17.18	11.7
Sending notifications	2.82	2.21
Total	21.91	15.21
Data read request class activities		
Forming a request	2.83	2.33
Request validation	3.83	2.92
Getting a projection	8.67	7.42
Converting a projection to a DTO	3	3.42
Total	18.33	16.08
Activities of the projection reconstruction process class		
Processing the request	4	4.43
Getting information about the projection	2.29	0
Getting a snapshot	5.86	6.57
Getting events	5	0
Building a Classical CQRS projection	36.43	19.29
Updating a projection in the database	4.86	4.14
Total	58.43	34.43
User data deletion activities in accordance with GDPR requirements		
Creating an event repository	12	0
Implementing the transformation function	11	0
Starting data migration	4	0
Anonymizing data in a snapshot	0	3
Anonymizing data in an event store	0	1
Rebuilding projections	16	4
Testing	8	3
Total	51	11
Event type version update operation activities		
Creating an event repository	12	0
Adding a new version of the event type	13	0
Implementing the transformation function	15	0
Starting data migration	4	0
Updating event type version in snapshot	0	8
Rebuilding projections	16	5
Testing	18	7
Total	63	20

Table 15

Total complexity

Process	Classical CQRS		mCQRS	
	Development	Modification	Development	Modification
Create commands set up	27.68	26.18	27.35	25.10
Update commands	42.04	33.54	34.00	30.50
Achieving consistency	20.26	14.50	15.50	13.50
Data read request	18.00	17.50	17.80	17.30
Projection rebuild rearrangements	38.83	33.33	28.69	26.69
Deleting user data in accordance with GDPR requirements	25.14	–	16	–
Event type version update	42.04	–	13.87	–

Table 16

Average server response time

Variation	Read requests, ms	Write requests, ms
Classical CQRS	43	132
mCQRS	43	210

Table 17

Performance rating scale based on average server response time

Satisfaction level	Unsatisfactorily	Minimally	Reasonably	Good	Perfectly
Threshold, ms	< 400	< 350	< 200	< 100	< 50
Score	1	2	3	4	5

Thus, the performance score for mCQRS is 2.93, and for Classical CQRS – 3.68.

The values of the metrics of the quality attribute components are normalized relative to each other and a matrix (10) is formed. The correlation of the normalized metrics is given in Fig. 7 and in Table 18:

$$A=\begin{pmatrix} m_{QAC_1}\ m_{QAC_2}\ldots m_{QAC_n} \\ c_{QAC_1}\ c_{QAC_2}\ \ldots\ c_{QAC_n} \end{pmatrix},$$

$$m_{QAC_i}+c_{QAC_i}=1,\ \ i\in\left[1;n\right], \qquad (10)$$

where m_{QAC} and c_{QAC} are normalized metrics of quality attribute components for the evaluated variations.

From Table 18 it is seen that the weights of individual processes, in particular setup commands and read requests, are almost the same. For other processes, the difference between the variations was about 0.13–0.15. At the same time, the indicators corresponding to the components of the problem space were characterized by significant deviations – 0.50 and 0.22. This result is justified by the fact that the mCQRS variation is focused on reducing the complexity of solving tasks in the CQRS-ES problem space.

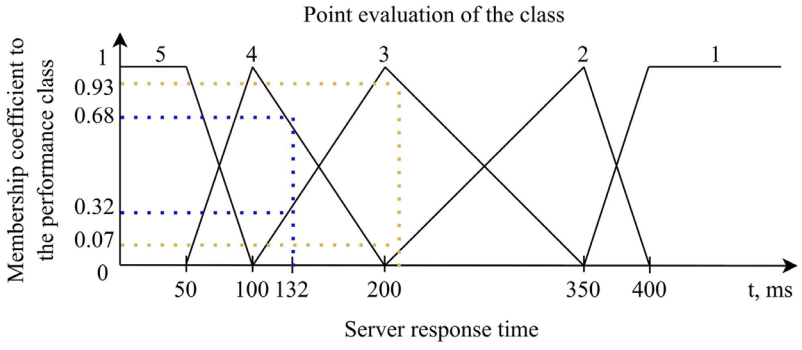

Fig. 6. Graphical representation of performance metric calculation

Table 18

Comparison of normalized variation metrics

QA component	Classical CQRS	mCQRS	Difference
Complexity			
Create setup commands (development)	0.4970	0.5030	0.0060
Create setup commands (modification)	0.4895	0.5105	0.0211
Update commands (development)	0.4471	0.5529	0.1057
Update commands (modification)	0.4763	0.5237	0.0475
Achieving consistency (development)	0.4334	0.5666	0.1331
Achieving consistency (modification)	0.4821	0.5179	0.0357
Data read requests (development)	0.4972	0.5028	0.0056
Data read requests (modification)	0.4971	0.5029	0.0057
Rebuilding projections (development)	0.4249	0.5751	0.1502
Rebuilding projections (modification)	0.4447	0.5553	0.1106
Update event type version	0.2481	0.7519	0.5038
Delete user data according to GDPR requirements	0.3889	0.6111	0.2222
Performance			
Data read operations	0.5000	0.5000	0
Data write operations	0.5567	0.4433	0.1135

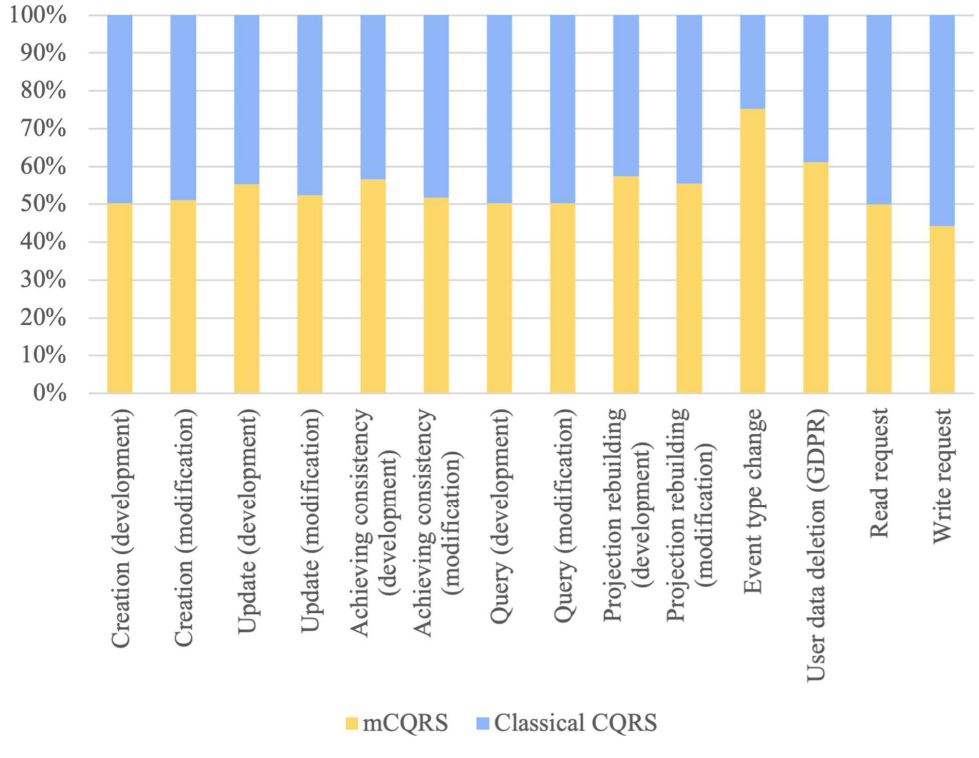

Fig. 7. Comparison of normalized variation metrics

Based on the parametric model, the weight vector w (11) is formed:

$$w=\begin{pmatrix} w_{QAC_1} \\ w_{QAC_2} \\ \vdots \\ w_{QAC_n} \end{pmatrix},$$

$$\sum_{i=1}^{n} w_{QAC_i}=1, \quad (11)$$

where w_{QAC} is the normalized weight of the quality attribute component; n is the number of components.

According to formulae (12) to (14) for the analytic hierarchy process (AHP) method [42], the relative estimates of variations were calculated:

$$r=Aw, \quad (12)$$

$$r=\begin{pmatrix} w_{QAC_1}\cdot m_{QAC_1}+\ldots+w_{QAC_n}\cdot m_{QAC_n} \\ w_{QAC_1}\cdot c_{QAC_1}+\ldots+w_{QAC_n}\cdot c_{QAC_n} \end{pmatrix}=\begin{pmatrix} r_{mCQRS} \\ r_{ClassicalCQRS} \end{pmatrix}, \quad (13)$$

$$r_{mCQRS}+r_{ClassicalCQRS}=1, \quad (14)$$

where r_{mCQRS} and $r_{ClassicalCQRS}$ are relative estimates of the applicability of the evaluated variations for the system described by the parametric model.

The r_{mCQRS} and $r_{ClassicalCQRS}$ values depend on two quality attributes (complexity and performance), each of which is represented by a set of components (Table 18). The relative estimates of variations were calculated separately, taking into account only the quality attribute of complexity: $r_{mCQRS}^{complexity}$ and $r_{ClassicalCQRS}^{complexity}$. The obtained values were used to calculate the predicted performance gain (ip_{mCQRS}) according to formula (15):

$$ip_{mCQRS}=\left(\frac{r_{mCQRS}^{complexity}}{r_{ClassicalCQRS}^{complexity}}-1\right)\cdot 100,$$

$$r_{mCQRS}^{complexity}>r_{ClassicalCQRS}^{complexity}. \quad (15)$$

For the average and marginal values of the parametric model, the results of applying the DSAV-CQRS-ES technology to support decision-making regarding the choice of a CQRSES variation, as well as the predicted performance gain, are summarized in Table 19.

The divergence of values is estimated using the standard deviation (s) and the coefficient of variation (Cv), calculated using formulae (5) to (7).

The time spent on applying the DSAV-CQRSES technology was measured separately for each stage and is given in Table 20.

In our work, instead of RTP, a parametric model implementation is used, and the requirements for RTP accuracy are lower than for the model implementation. Therefore, when estimating the RTP development time, only the first two aggregates implemented within the parametric model are taken into account.

Table 19

Results of DSAV-CQRSES technology application

Model parameter values	Ease of development / performance	Projected productivity growth, %	Classical CQRS, %	mCQRS, %
Minimum	90 / 10	28.15	44.51	55,49
	80 / 20		45.18	54,82
	50 / 50		47.20	52,80
Average	90 / 10	34.36	43.46	56,54
	80 / 20		44.25	55,75
	50 / 50		46.62	53,38
Maximum	90 / 10	38.89	42.73	57,27
	80 / 20		43.61	56,39
	50 / 50		46.22	53,78
Standard deviation (s)			1,54	1,54
Coefficient of variation (Cv), %			3,44	2,8

Table 20

Duration of application of DSAV-CQRSES technology

Activity	Number of specialists	Duration, hours	Labor intensity, person-hours
Preparation of architectural candidate solutions	1	8	8
Introduction to DSAV-CQRSES technology		4	4
Complexity calculation		4	4
RTP development		21	21
Performance measurement		1	1
Calculate overall complexity and performance		1	1
Calculate recommended solution score and predicted performance gain		1	1
Total		40	40

5. 3. Comparison of software architecture evaluation methods

The results of the comparison of ATAM and DSAV-CQRS-ES technology are summarized in comparative Table 21.

Table 21

Results of comparing ATAM and DSAV-CQRSES technology

Parameter	**ATAM**	DSAV-CQRSES
Number of stages	9 actions within 2 phases	12 actions within 2 phases
Application time	4 specialists involved; duration – 32 hours; labor intensity – 88 person-hours	1 specialist involved; duration – 40 hours; labor intensity – 40 person-hours
Starting artifacts	System design documentation with identified risks, sensitivity points, and compromise points	System design documentation; evaluation of recommended solution; evaluation of predicted performance gains; RTP
Calculation automation	Information on the automation of calculations is not provided in the available sources. The use of language models to support the evaluation process can be considered as a promising direction	Automated calculation of overall complexity and performance; data preparation and MAI execution; calculation of predicted performance gains
Epistemic uncertainty of choice	Three teams chose mCQRS; one chose Classical CQRS; another was hesitant at first, but eventually chose Classical CQRS as well	The mCQRS variation is recommended for both average and extreme values of the parametric model
Disparity of outcomes	The coefficient of variation for the number of sensitivity points, compromise points, and risks considered is 47.14% for mCQRS and 33.64% for Classical CQRS	The coefficient of variation for the recommendation to use mCQRS is 2.8%, and for Classical CQRS it is 3.44%

The comparison showed that although the DSAV-CQRSES application takes about 8 hours longer than ATAM, the total labor intensity was 2.2 times less since the procedure is performed by one person. At the same time, to increase objectivity within the team, it is advisable to add a short discussion of the results and making a joint decision based on them (in our case, about 2 hours for 4 specialists).

When comparing the initial artifacts, the presence of RTP was noted among the materials generated by the DSAV-CQRSES technology. RTP serves as the basis for further design and development of the system and reduces complexity at the initial stage.

The reproducibility of the results was also assessed. The DSAV-CQRSES technology, which relies on quantitative metrics, recommended the mCQRS variation in 100% of cases, while ATAM, which is based on expert assessment, provided the correct choice only in 60% of cases.

There is no direct evidence of the influence of participant experience, but errors in the application of ATAM occurred in teams with predominantly less experienced specialists. This may indicate higher requirements for the competence of performers to reliably apply ATAM for the task of comparing CQRSES variations.

Feedback by specialists from teams that made errors in the application of ATAM:

– Specialist 1 (Team 1). During the application of ATAM, they concluded that mCQRS might not meet the performance requirements due to possible delays in updating the event store and the state snapshot database, so they decided to use Classical CQRS. However, further measurements showed that these concerns were unfounded: the system met the SLA performance requirements;

– Specialist 3 (Team 1). The estimated indicators of the system's flexibility and modifiability turned out to be insufficiently accurate due to the high uncertainty of the input data; more detailed metrics are required to correctly distinguish variations;

– Specialist 13 (Team 5). Significant doubts were raised about the implementation of synchronization of dependent events. Without the elaboration of RTP and the specification of the projection reconstruction mechanism, the participants assumed that in mCQRS, due to the lack of an explicit mechanism for replaying events (replay), the implementation would be significantly more complicated than in Classical CQRS. In the end, for performance reasons, we preferred Classical CQRS.

6. Discussion of results based on comparing the software architecture evaluation methods

Our results show that for the task of choosing between CQRSES architecture variations, the specialized DSAV-CQRSES technology provides more consistent and reproducible results than the general-purpose ATAM method.

Firstly, this is explained by the difference in the level of formalization of the two approaches. ATAM involves building a utility tree, identifying scenarios, sensitivity points, trade-off points, and risks based on expert analysis. The corresponding stages are shown in Fig. 3–5 and Tables 6–10. This procedure is flexible, but its result significantly depends on the experience of the participants, the completeness of the considered scenarios, and the way of interpreting architectural risks. In the experiment, this was manifested in the fact that different teams, working with the same input data, provided different recommendations for choosing an architectural variation.

Secondly, general-purpose methods are not tied to specific architectural paradigms and therefore take into account the specifics of CQRSES to a limited extent. In contrast, DSAV-CQRSES is purposefully focused on a narrow class of systems (CQRSES): the method is not universal, but it provides higher consistency of results in this context. This is confirmed by the range of results (Table 21). In particular, the coefficient of variation calculated from formulae (5) to (7) for mCQRS based on the results of ATAM application is 33.64% (Table 11), while for DSAV-CQRSES it is 2.8% (Table 19).

Third, the level of abstraction in general-purpose methods makes it difficult to distinguish nuances critical to variations – for example, working with state snapshots, event replay strategies, synchronization of dependent events. In DSAV-CQRSES, these parameters are embedded in the evaluation parameters.

Fourth, ATAM relies heavily on expert judgment (Tables 6, 8), which increases the dependence on the human factor and increases the divergence of results. DSAV-CQRS-ES uses quantitative metrics (Tables 13–18 and formulae (8) to (15)), which reduces epistemic uncertainty and improves reproducibility (Epistemic Choice Uncertainty, Table 21).

In addition, the difference between the methods was manifested not only in the final choice of architectural variation but also in the complexity of the application. ATAM required the participation of several specialists and group sessions (Table 12), while DSAV-CQRSES was applied by one specialist using formalized calculations (Table 20). Therefore, despite the longer calendar duration of individual stages of DSAV-CQRSES, the total labor costs were lower.

Compared to similar studies [12, 13], the proposed method comparison approach involves substantiating the adequacy of input data for the experiment and involving specialists with at least three years of experience in software development in its conduct. This increases the practical relevance of the conclusions obtained. In addition, unlike [6–9], this approach provides the possibility of quantitatively reflecting the advantages and limitations of the studied methods. The representation of results is structurally similar to [7] but, in contrast, is supplemented by quantitative indicators of the application of methods in a given project context.

The main limitations of our study include, first of all, the context of its application. The results relate to architectural variations of CQRSES. Accordingly, the adequacy of the conclusions was confirmed primarily within the studied class of architectural solutions. Another limitation is the number of applications of the ATAM method within the experiment. This is due to the high complexity of such assessments, in particular, the need to involve a significant number of specialists and significant time costs for the analysis of test systems.

A certain drawback of the study is the lack of a mechanism for summarizing the results to a single integral indicator. In its current form, the results are represented in the form of a table of individual characteristics. Further development of this study may involve eliminating the specified drawback by integrating multi-criteria analysis methods. However, the implementation of such an approach is associated with certain difficulties, in particular, with the lack of clarity in determining the weight coefficients.

7. Conclusions

1. A parametric model has been built for a class of systems for which the optimal variation is mCQRS; its adequacy has been experimentally substantiated. A feature of the model is the combination of formalized classes of use cases with system and contextual parameters that characterize both the structure of the computer system and the conditions of its development and maintenance. Unlike the description of a separate test case, the model specifies a generalized class of real systems through intervals of parameter values.

2. A series of experiments to determine the optimal variation of the CQRSES architecture using ATAM and DSAV-CQRSES technology showed that according to the results of five ATAM applications, the accuracy was 60%, the coefficient of variation of the results was 33.64%, and the total labor intensity was 88 person-hours. According to the results of nine DSAV-CQRSES applications, the accuracy was 100%, the coefficient of variation of the results was 2.8% (which is 12 times less), and the total labor intensity was 40 person-hours (2.2 times less).

The key factor in reducing epistemic uncertainty and increasing objectivity when applying DSAV-CQRSES is the absence of expert assessment as a component of the comparison procedure. Thus, by the set of indicators of accuracy, reproducibility, and labor intensity, DSAV-CQRSES demonstrates higher objectivity of decision-making support when choosing the optimal variation of CQRSES.

3. Using the proposed methodology, a comparison of ATAM to DSAV-CQRSES was carried out. The advantage of the methodology is the combination of a qualitative comparison of the methods with a quantitative assessment of their application according to the following parameters: the level of epistemic uncertainty, application time, number of stages, output artifacts, the degree of automation of calculations and reproducibility of results. The methodology also includes substantiation of the correctness of the reference architectural variation and involvement of qualified specialists for experimental application of candidate methods. The results of our study indicate that the applied methodology allows for objective comparison of approaches to software architecture assessment within the class of projects corresponding to the parametric model.

Conflicts of interest

The authors declare that they have no conflicts of interest in relation to the current study, including financial, personal, authorship, or any other, that could affect the study and the results reported in this paper.

Financing

The research was partially funded by the Ministry of Foreign Affairs of the Czech Republic under project 25-PKVV-UM-004 entitled “Development of Education, Research, and International Cooperation at Oles Honchar Dnipro National University (DNU)” implemented by Charles University and DNU.

Within the scope of the project, a use-case classification approach was developed and applied to architectural variations.

Data availability

Manuscript has associated data in a data repository:

– presentation “Choose the optimal variation of CQRS with ES architecture with ATAM”. Access mode: https://doi.org/10.5281/zenodo.18472924;

– dataset “Parametric model implementation time for Classical CQRS and mCQRS”. Access mode: https://doi.org/10.5281/zenodo.18472975;

– dataset “Calculation of cognitive functional complexity of activities for processes in the CQRS with Event sourcing systems”. Access mode: https://doi.org/10.5281/zenodo.18509043;

– dataset “Classification of Use Cases in Systems Based on CQRS with Event Sourcing”. Access mode: https://doi.org/10.5281/zenodo.18621093;

– software “Monorepo with mCQRS and Classical CQRS implementation for parametric model”. Access mode: https://doi.org/10.5281/zenodo.18473292.

Use of artificial intelligence

AI tool was used:
– Model and number of AI: ChatGPT 5.2;
– Where exactly it was used? - The entire manuscript;
– What exactly was done using AI tools?
– identification and correction of grammatical and punctuation errors, in parallel with other tools (e.g., Reverso Context);
– finding sources for a literature review;
– How the authors checked the results provided by AI tools;
– the authors reviewed the suggested grammar and punctuation edits and applied them selectively;
– the authors assessed the relevance of the suggested sources by reading the recommended articles;
– result provided by the AI tool did not influence the conclusions of the study.

Acknowledgments

We sincerely appreciate DBBSoftware company [43] 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.

Authors' contributions

Dmytro Hruzin: Conceptualization, Methodology, Software, Validation, Investigation, Data Curation, Writing – original draft, Writing – review & editing, Supervision, Project administration; **Oleksandr Lytvynov**: Conceptualization, Writing – review & editing.

References

1. Betts, D., Dominguez, J., Melnik, G., Simonazzi, F., Subramanian, M. (2013). Exploring CQRS and Event Sourcing: A Journey into High Scalability, Availability, and Maintainability with Windows Azure. Microsoft patterns & practices, 376. Available at: https://download.microsoft.com/download/e/a/8/ea8c6e1f-01d8-43ba-992b-35cfcaa4fae3/cqrs_journey_guide.pdf
2. CQRS Documents by Greg Young. Available at: https://cqrs.files.wordpress.com/2010/11/cqrs_documents.pdf
3. Breivold, H. P., Crnkovic, I., Larsson, M. (2012). A systematic review of software architecture evolution research. Information and Software Technology, 54 (1), 16–40. https://doi.org/10.1016/j.infsof.2011.06.002
4. Ford, N., Parsons, R., Kua, P., Sadalage, P. (2022). Building Evolutionary Architectures. O'Reilly Media, 262. Available at: https://www.oreilly.com/library/view/building-evolutionary-architectures/9781492097532/
5. Soliman, M., Avgeriou, P., Li, Y. (2021). Architectural design decisions that incur technical debt – An industrial case study. Information and Software Technology, 139, 106669. https://doi.org/10.1016/j.infsof.2021.106669
6. Sobhy, D., Bahsoon, R., Minku, L., Kazman, R. (2021). Evaluation of Software Architectures under Uncertainty. ACM Transactions on Software Engineering and Methodology, 30 (4), 1–50. https://doi.org/10.1145/3464305
7. Babar, M. A., Zhu, L., Jeffery, R. (2004). A framework for classifying and comparing software architecture evaluation methods. 2004 Australian Software Engineering Conference. Proceedings, 309–318. https://doi.org/10.1109/aswec.2004.1290484
8. Roy, B., Graham, N. (2008). Methods for Evaluating Software Architecture: A Survey. Technical Report No. 2008-545. School of Computing Queen's University at Kingston Ontario. Available at: https://research.cs.queensu.ca/TechReports/Reports/2008-545.pdf?utm_source=chatgpt.com
9. Mattsson, M., Grahn, H., Mårtensson, F. (2006). Software Architecture Evaluation Methods for Performance, Maintainability, Testability, and Portability. Presented at the Second International Conference on the Quality of Software Architectures (QoSA 2006). Västerås. Available at: https://www.researchgate.net/publication/200622064_Software_Architecture_Evaluation_Methods_for_Performance_Maintainability_Testability_and_Portability
10. Kazman, R., Klein, M., Clements, P. (2000). ATAM: Method for Architecture Evaluation. Technical report: CMU/SEI-2000-TR-004, ESC-TR-2000-004. Pittsburgh: Product Line Systems, 71. Available at: https://www.researchgate.net/publication/2814191_ATAM_Method_for_Architecture_Evaluation
11. Lytvynov, O., Hruzin, D. (2025). Decision-making on Command Query Responsibility Segregation with Event Sourcing architectural variations. Technology Audit and Production Reserves, 4 (2 (84)), 37–59. https://doi.org/10.15587/2706-5448.2025.337168
12. Abrahão, S., Insfran, E. (2017). Evaluating Software Architecture Evaluation Methods. Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering, 144–153. https://doi.org/10.1145/3084226.3084253
13. Gonzalez-Huerta, J., Insfran, E., Abrahão, S., Scanniello, G. (2015). Validating a model-driven software architecture evaluation and improvement method: A family of experiments. Information and Software Technology, 57, 405–429. https://doi.org/10.1016/j.infsof.2014.05.018
14. Eloranta, V.-P., van Heesch, U., Avgeriou, P., Harrison, N., Koskimies, K. (2014). Lightweight Evaluation of Software Architecture Decisions. Relating System Quality and Software Architecture, 157–179. https://doi.org/10.1016/b978-0-12-417009-4.00006-5
15. Babar, M. A., Kitchenham, B. (2007). Assessment of a Framework for Comparing Software Architecture Analysis Methods. Electronic Workshops in Computing. https://doi.org/10.14236/ewic/ease2007.2
16. Babar, M. A., Gorton, I. (2004). Comparison of Scenario-Based Software Architecture Evaluation Methods. 11th Asia-Pacific Software Engineering Conference, 600–607. https://doi.org/10.1109/apsec.2004.38

17. Hruzin, D. L., Lytvynov, O. A. (2025). The impact of quality attribute selection on comparing CQRS with event sourcing architectural variations. Proceding of the XVIII International Scientific and Practical Conference. Odesa, 614–616. Available at: https://if.uu.edu.ua/wp-content/uploads/2025/11/zbirnyk-tez-konferentsii-itia-2025.pdf
18. Le Thi Kim Nhung, H., Van Hai, V., Hoc, H. T. (2021). Analyzing Correlation of the Relationship Between Technical Complexity Factors and Environmental Complexity Factors for Software Development Effort Estimation. Software Engineering Application in Informatics, 835–848. https://doi.org/10.1007/978-3-030-90318-3_65
19. da Silva, A. R., Savić, D., Vlajić, S., Antović, I., Lazarević, S., Stanojević, V., Milić, M. (2015). A pattern language for use cases specification. Proceedings of the 20th European Conference on Pattern Languages of Programs, 1–18. https://doi.org/10.1145/2855321.2855330
20. Minsky, M. (1974). A Framework for Representing Knowledge. MIT Research Lab Technical Report. Massachusetts Institute of Technology, Cambridge, MA, United States. Available at: https://courses.media.mit.edu/2004spring/mas966/Minsky%201974%20Framework%20for%20knowledge.pdf
21. Hruzin, D. (2026). Classification of Use Cases in Systems Based on CQRS with Event Sourcing. Zenodo. https://doi.org/10.5281/zenodo.18621093
22. Janssen, P., Stouffs, R. (2015). Types of Parametric Modelling. Proceedings of the 20th Conference on Computer Aided Architectural Design Research in Asia (CAADRIA), 157–166. https://doi.org/10.52842/conf.caadria.2015.157
23. Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. The Annals of Statistics, 29 (5). https://doi.org/10.1214/aos/1013203451
24. Kohl, M., Ruckdeschel, P. (2010). R Package distrMod: S4 Classes and Methods for Probability Models. Journal of Statistical Software, 35 (10). https://doi.org/10.18637/jss.v035.i10
25. General Data Protection Regulation. Available at: https://gdpr-info.eu/
26. Aljoumah, E., Al-Mousawi, F., Ahmad, I., Al-Shammri, M., Al-Jady, Z. (2015). SLA in Cloud Computing Architectures: A Comprehensive Study. International Journal of Grid and Distributed Computing, 8 (5), 7–32. https://doi.org/10.14257/ijgdc.2015.8.5.02
27. Application Performance Index – Apdex Final Technical Specification (2005). Apdex Alliance. Available at: https://www.apdex.org/wp-content/uploads/2020/09/Apdex_Technical_Specification.pdf
28. Tsyganok, I. (2016). Pair-Programming from a Beginner's Perspective. Agile Processes, in Software Engineering, and Extreme Programming, 270–277. https://doi.org/10.1007/978-3-319-33515-5_25
29. Hruzin, D. (2026). dmytrohruzin/DSAV-CQRSES-ATAM-parametic-model: TODOs for MVP (1.0.2). Zenodo. https://doi.org/10.5281/zenodo.18473292
30. DSAV-CQRSES-ATAM-parametic-model. Available at: https://sonarcloud.io/summary/overall?id=dmytrohruzin_DSAV-CQRSES-ATAM-parametic-model&branch=main
31. CQRS pattern. Available at: https://learn.microsoft.com/en-us/azure/architecture/patterns/cqrs
32. Event Sourcing pattern. Available at: https://learn.microsoft.com/en-us/azure/architecture/patterns/event-sourcing
33. Fowler, M. (2011). CQRS. Available at: https://martinfowler.com/bliki/CQRS.html
34. Di Pompeo, D., Tucci, M. (2023). Quality Attributes Optimization of Software Architecture: Research Challenges and Directions. 2023 IEEE 20th International Conference on Software Architecture Companion (ICSA-C), 252–255. https://doi.org/10.1109/icsa-c57050.2023.00061
35. Hruzin, D. (2026). Presentation: Choose the optimal variation of CQRS with ES architecture with ATAM. https://doi.org/10.5281/zenodo.18472924
36. Taguchi, G., Chowdhury, S., Wu, Y. (2004). Taguchi's Quality Engineering Handbook. Wiley. https://doi.org/10.1002/9780470258354
37. Lytvynov, O., Khandetskyi, V., Lytvynov, M. (2025). The Estimation of Effort for Domain-Driven Architectural Variations Migration using Modified Use Case Size Points Method. Proceding of the 1st International Scientific and Practical Conference (MIT@AIS'2025s). Kharkiv-Yaremche, 149–152. Available at: https://drive.usercontent.google.com/u/0/uc?id=1iVEiy8u0IJ9jgcQyvyhRiId3vKpAyGf-
38. Hruzin, D. (2026). Calculation of cognitive functional complexity of activities for processes in the CQRS with Event sourcing systems. Zenodo. https://doi.org/10.5281/zenodo.18509043
39. Wang, Y., Shao, J. (2003). Measurement of the cognitive functional complexity of software. The Second IEEE International Conference on Cognitive Informatics, 2003. Proceedings, 67–74. https://doi.org/10.1109/coginf.2003.1225955
40. Ionita, M. T., America, P., Hammer, D. K., Obbink, H., Trienekens, J. J. M. A scenario-driven approach for value, risk, and cost analysis in system architecting for innovation. Proceedings. Fourth Working IEEE/IFIP Conference on Software Architecture (WICSA 2004), 277–280. https://doi.org/10.1109/wicsa.2004.1310709
41. Zadeh, L. A. (1965). Fuzzy sets. Information and Control, 8 (3), 338–353. https://doi.org/10.1016/s0019-9958(65)90241-x
42. Brunelli, M. (2015). Introduction to the Analytic Hierarchy Process. In (Editor), SpringerBriefs in Operations Research. Springer International Publishing. https://doi.org/10.1007/978-3-319-12502-2
43. DBBSoftware. Available at: https://dbbsoftware.com/