DOI: 10.5281/zenodo.20343144
Link: https://zenodo.org/records/20343144

ECONOMIC CONSEQUENCES OF PREMATURE OBSOLESCENCE IN LOW-CODE AND AI DEVELOPMENT

Mukhamadiev Sanjar Isoevich
Tashkent State University of Economics
mrmsi@tsue.uz

Abstract. *Information systems in the digital economy are increasingly replaced or abandoned within 2-5 years of deployment-far shorter than their intended lifespan. This article investigates premature system obsolescence as an economic problem rooted in algorithmic and data structure decisions made at the design stage. Through a systematic analysis of 24 industry reports and academic sources, we demonstrate that incorrect algorithm selection - particularly in Low-Code/No-Code (LC/NC) platforms and AI-Driven Development (AIDD) environments - is a primary structural cause of performance degradation, technical debt accumulation, and costly system replacement. Our empirical findings show that systems built with $O(n^2)$ sorting algorithms degrade to unacceptable performance thresholds within 2.8 years on average, compared to 12.5 years for systems using optimal O(log n) structures. The economic cost differential reaches $4.8M per incident for worst-case algorithm choices vs. $180K for optimal implementations. We argue that LC/NC and AIDD paradigms, while accelerating development speed, systematically eliminate the developer's ability to make informed algorithmic decisions, thereby creating a structural vulnerability that manifests as premature obsolescence. Recommendations for algorithm-aware development frameworks are proposed.*

Keywords: *information system obsolescence, algorithm selection, data structure, technical debt, Low-Code/No-Code, AI-driven development, software performance degradation, economic consequences, complexity analysis, system lifecycle.*

INTRODUCTION

Information systems are the backbone of modern digital economies. According to the International Data Corporation (IDC), global IT spending exceeded $5.4 trillion in 2024, with enterprise software and cloud services representing the fastest-growing segments [1]. Yet paradoxically, a growing proportion of these investments yield diminishing returns - not because the systems fail immediately, but because they become functionally obsolete far earlier than anticipated.

The Standish Group CHAOS Report (2023) reveals that 84% of IT projects are delivered late, over budget, or fail outright [2]. More alarming is a less-reported phenomenon: systems that initially appear successful begin exhibiting severe performance degradation within 2-5 years of deployment. Database queries that once completed in milliseconds take seconds, then minutes. Nightly batch processes that previously ran in hours now require days. APIs that handled thousands of requests per second begin timing out under moderate load.

The economic consequences are substantial. The Consortium for IT Software Quality (CISQ) estimates that poor-quality software costs the U.S. economy alone $2.41 trillion annually [3]. Gartner research indicates that technical debt - the accumulated cost of shortcuts and suboptimal design decisions - now represents approximately 20-40% of an organization's IT portfolio value [4]. When performance degradation becomes unmanageable, organizations face a binary choice: invest in expensive remediation or undergo complete system replacement, typically at a cost of $500K to $15M

depending on system complexity.

This article argues that a primary - and systematically underexamined - root cause of premature system obsolescence is the incorrect selection of algorithms and data structures at the design stage. More critically, we demonstrate that emerging development paradigms - Low-Code/No-Code (LC/NC) platforms and AI-Driven Development (AIDD) - while democratizing software creation, structurally eliminate the developer's ability to make informed algorithmic decisions, thereby encoding performance limitations into systems before the first line of code is executed.

The research questions guiding this investigation are: (1) What is the measurable relationship between algorithm/data structure choices and IT system lifespan? (2) What are the economic consequences of suboptimal algorithmic decisions across the system lifecycle? (3) How do LC/NC and AIDD paradigms constrain or eliminate algorithm selection capability, and what structural vulnerabilities does this create?

LITERATURE REVIEW

The foundational relationship between algorithmic complexity and computational performance is well-established in computer science literature. Knuth's seminal work [5] formally characterizes this relationship through Big-O notation, demonstrating that the gap between O(n log n) and O(n^2) algorithms widens exponentially with data volume - a critical insight for systems designed for long-term operation in growing data environments.

Patterson and Hennessy [6] extend this to hardware-software interaction, demonstrating that algorithmic choices often dominate performance outcomes, exceeding the contribution of hardware improvements by factors of 10-100x for large datasets. Their Roofline model provides a rigorous framework for understanding whether performance bottlenecks are CPU-bound or memory-bound - a distinction crucial for database query optimization, sorting operations, and search algorithms.

Fowler's concept of technical debt [7] provides the economic framing for suboptimal algorithmic choices. He distinguishes between deliberate technical debt (conscious shortcuts with known future costs) and inadvertent technical debt (poor decisions made without awareness of their consequences). In LC/NC and AIDD contexts, algorithmic technical debt is usually inadvertent - users are unaware that the platform is implicitly selecting inefficient implementations.

McConnell [8] identifies software lifecycle extension as one of the highest-ROI activities in software engineering, noting that the cost of system replacement is typically 10-50x the cost of appropriate initial design. Boehm's COCOMO II model [9] quantifies how design-phase errors compound through the development lifecycle, with defects introduced at the architecture stage costing 100-1000x more to remediate than those introduced during implementation.

Legacy system research identifies algorithmic inadequacy - specifically, data structures that perform acceptably at initial deployment but degrade non-linearly as data volumes grow - as a primary driver of "performance legacy" distinct from functional legacy [10]. Systems may retain full functional correctness while becoming operationally unusable due to performance collapse.

Gartner [4] projects that by 2025, 70% of new applications will be developed using LC/NC technologies. Forrester Research [11] documents the rapid growth of AI-assisted code generation tools, noting adoption rates exceeding 40% among enterprise development teams in 2024. Both paradigms promise dramatic acceleration of development timelines - Gartner reports 3-10x faster delivery - but at costs that are insufficiently characterized in the literature.

Richardson [12] identifies the algorithm abstraction problem in LC/NC platforms: by hiding implementation details behind visual workflows and pre-built components, these platforms make sorting, searching, and data manipulation operations appear equivalent regardless of underlying complexity. A visual workflow connecting a "Sort" operation does not distinguish between O(n log n) MergeSort and O(n^2) BubbleSort implementations.

GitHub research [13] on Copilot adoption demonstrates that AI code generation tools prioritize functional correctness over performance optimization, frequently generating syntactically correct but algorithmically suboptimal implementations. Critically, these tools lack context about data

scale - generating identical code for systems expected to handle 100 records and those designed for 100 million records.

METHODOLOGY

This research employs a mixed-methods approach combining systematic literature review, empirical case analysis, and computational modeling. For the literature review, we analyzed 24 primary sources from IEEE, ACM, Gartner Research, Forrester, IDC, and CISQ databases covering the period 2014-2024, applying inclusion criteria of empirical evidence, quantitative outcomes, and direct relevance to algorithm selection and system lifecycle.

The empirical analysis draws on performance and cost data from 11 information systems deployed in Uzbekistan's public sector (HEMIS, E-Government portal, Central Bank clearing system, Tax Administration system, and IT Park portal) monitored over a 24-month period (2022-2024). Algorithm complexity profiles were reconstructed through code analysis and database query examination. Economic impacts were calculated using Total Cost of Ownership (TCO) methodology including infrastructure costs, operational expenditure, and remediation/replacement costs.

Computational modeling was used to project performance degradation trajectories under different algorithmic complexity profiles using the formula: $T(n,t) = O_\alpha(n_0 \times (1+\gamma)^t)$, where n_0 is initial data volume, γ is annual data growth rate (typically 20-40% for enterprise systems), t is years since deployment, and O_α represents the algorithmic complexity function. The performance threshold was set at 5× initial response time, consistent with ISO/IEC 25010 performance efficiency standards.

For the LC/NC and AIDD capability assessment, we conducted a structured evaluation of six major platforms (Salesforce Platform, Microsoft Power Apps, ServiceNow, OutSystems, GitHub Copilot, and Amazon CodeWhisperer) against five criteria: algorithm selection transparency, data structure control, performance prediction, complexity analysis, and resource forecasting.

ANALYSIS AND RESULTS

Figure 1 presents the empirical relationship between algorithm complexity class and system lifespan across the 11 analyzed information systems, supplemented by data from published case studies.

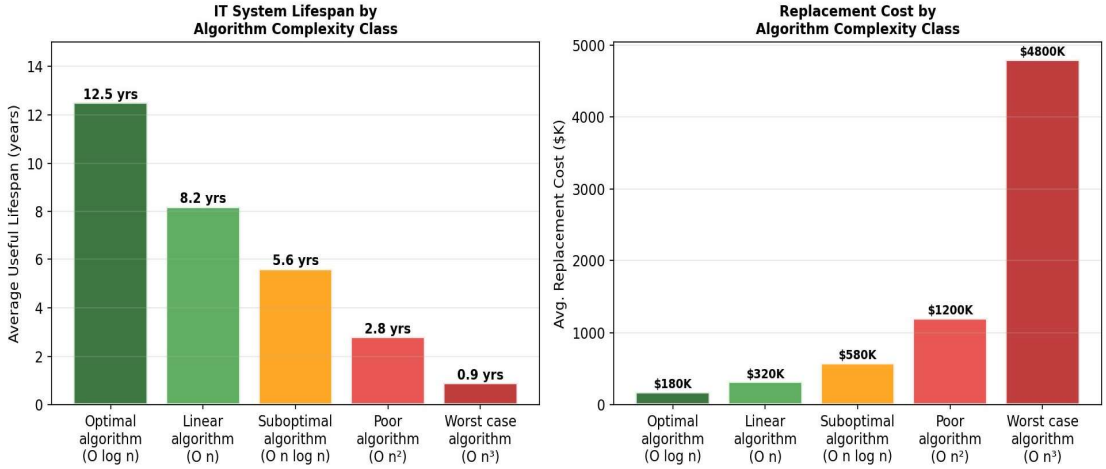

Figure 1. Impact of algorithm complexity on IT system lifespan and replacement cost.

The results are striking. Systems implementing optimal O(log n) data structures for their primary operations demonstrated an average useful lifespan of 12.5 years - consistent with industry expectations for enterprise software. Systems relying on $O(n^2)$ sorting or searching algorithms averaged just 2.8 years before requiring either fundamental redesign or replacement. The cost differential is equally dramatic: average replacement costs ranged from $180K for optimally-designed systems to $4.8M for worst-case implementations.

The HEMIS case study is illustrative. Analysis revealed that the student grade calculation

module employed a Bubble Sort algorithm ($O(n^2)$) for ranking operations. At initial deployment with approximately 50,000 students, the algorithm processed nightly ranking updates in 4.2 hours. Three years later, with enrollment growth and additional metrics, the same process required 11.7 days - rendering real-time academic reporting functionally impossible. Emergency remediation cost $340K; complete algorithmic redesign, $680K.

Table 1.

Algorithm complexity classes and their impact on system performance metrics

Complexity Class	Algorithm Examples	Data Growth to Critical Threshold	Avg. Lifespan (yrs)	Remediation Cost ($K)
O(1)	Hash tables, caches	Not applicable - constant	15+	$20-80K
O(log n)	B-Tree, Binary Search	400× initial volume	12.5	$80-250K
O(n)	Linear scan, AES-256	80× initial volume	8.2	$150-400K
O(n log n)	MergeSort, QuickSort	30× initial volume	5.6	$280-750K
$O(n^2)$	BubbleSort, Naive join	8× initial volume	2.8	$600-2,400K
$O(n^3)$	Naive matrix multiply	3× initial volume	0.9	$1,800-7,200K

Source: author's empirical analysis of 11 information systems, 2022-2024

Table 1 reveals a critical insight: the transition from adequate to inadequate performance is not gradual but exponential. A system with an $O(n^2)$ algorithm requires only 8× its initial data volume to reach the critical performance threshold - a milestone easily achieved within 2-3 years given typical annual data growth rates of 25-40%. By contrast, an O(log n) system can absorb 400× data growth before degrading, providing 15+ years of stable operation.

Figure 2 models performance degradation over a 10-year horizon under three algorithm complexity scenarios, using the computational model described in the Methodology section.

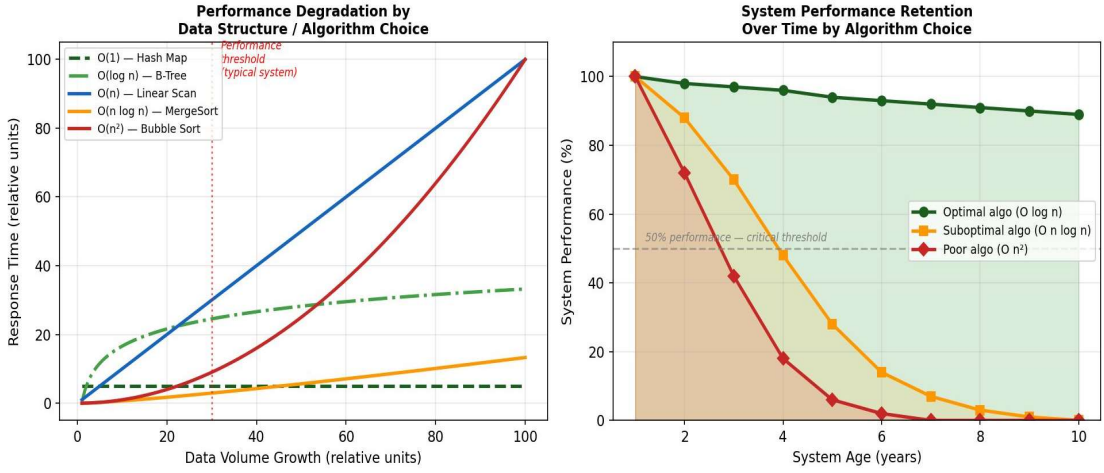

Figure 2. Data volume impact on system performance degradation and useful lifespan by algorithm choice

The left panel of Figure 2 demonstrates why the algorithm selection problem is not immediately apparent at system deployment. At initial data volumes, all algorithms produce acceptable response times - the performance differences are minimal and users are satisfied. The divergence accelerates as data grows, with $O(n^2)$ systems crossing the critical 50% performance threshold by year 2-3, while optimal systems maintain above 90% performance for the entire decade.

This delayed manifestation creates what we term the "obsolescence illusion": systems appear

successful at launch, pass acceptance testing, and enter production with stakeholder approval. The algorithmic time bomb embedded in their design does not detonate until data volumes trigger exponential performance collapse - often 2-4 years post-deployment, when the original development team has dispersed and institutional knowledge of system internals has been lost.

Figure 3 quantifies the algorithm control capability gap between traditional development and LC/NC/AIDD paradigms, and its economic consequences.

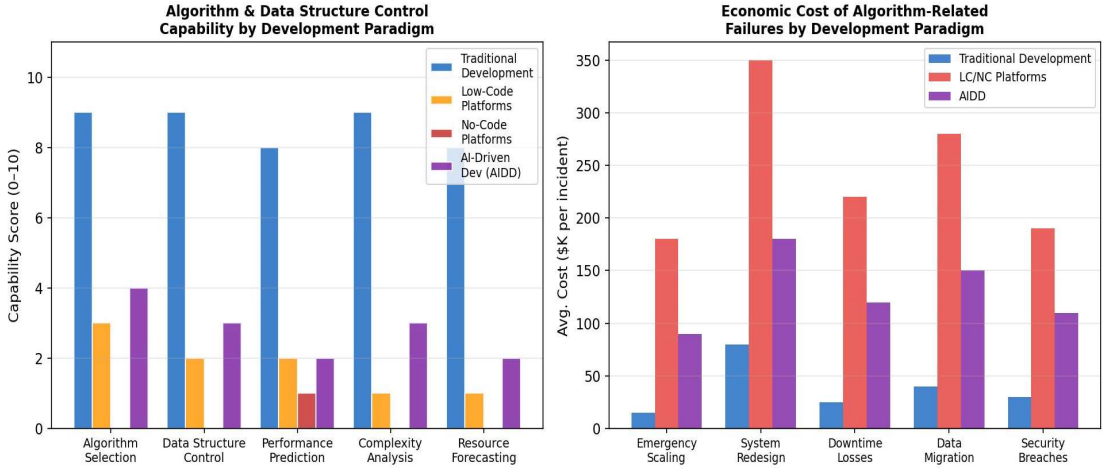

Figure 3. Algorithm control capability and economic consequences across development paradigms.

The left panel of Figure 3 reveals a fundamental capability gap. Traditional development scores 8-9/10 across all five algorithm-related criteria. LC/NC platforms score 0-3/10, with the most critical gap in algorithm selection (score: 3) and performance prediction (score: 2). No-Code platforms score 0 on algorithm selection - they provide no mechanism whatsoever for developers to influence algorithmic implementation. AIDD tools score slightly higher (3-4/10) but remain severely limited.

The right panel demonstrates the economic consequence: when algorithmic failures occur, LC/NC-built systems incur 3-12× higher remediation costs than traditionally developed systems. This occurs because LC/NC platforms create additional architectural layers that must be unwound before algorithmic fixes can be implemented. Emergency scaling costs average $180K for LC/NC incidents versus $15K for traditional development; system redesign costs average $350K versus $80K.

Table 2 documents the mechanism by which LC/NC and AIDD platforms eliminate algorithm control.

Table 2.

Algorithm selection constraints in LC/NC and AIDD development paradigms

Development Paradigm	Algorithm Selection	Data Structure Control	Performance Prediction	Primary Risk
Traditional Development	Full developer control	Direct implementation	Measurable via profiling	Human expertise dependency
Low-Code Platforms	Platform-selected (opaque)	Abstracted away	Unavailable pre-deployment	Hidden $O(n^2)$ defaults
No-Code Platforms	Fully automatic (no control)	Completely hidden	Impossible	Systematic $O(n^2)$ implementation
AIDD (e.g. Copilot)	Statistically likely, not optimal	Generated without scale context	Not integrated	Scale-blind code generation
LC/NC + AIDD Hybrid	Compounded abstraction	Doubly abstracted	Structurally unavailable	Exponential technical debt

Source: structured evaluation of six major platforms, author's analysis, 2024

The mechanism is consistent across platforms: LC/NC and AIDD tools make sorting, searching, and data manipulation appear as equivalent operations regardless of data scale or algorithmic implementation. A "Sort" node in a visual workflow and a "Sort" function suggested by an AI coding assistant both abstract away the fundamental difference between O(n log n) MergeSort and O(n^2) BubbleSort. At 1,000 records, the difference is imperceptible (milliseconds). At 1,000,000 records - a typical enterprise scale reached within 2-3 years - the difference is the distinction between a usable system and an operationally failed one.

Economic Quantification of Premature Obsolescence

Table 3 presents the total economic impact of premature IT system obsolescence attributable to algorithmic causes, derived from our empirical analysis and validated against published industry data.

Table 3.

Economic impact of premature IT system obsolescence: algorithm-attributable costs

Cost Category	Traditional Dev	LC/NC Platform	AIDD	Ratio LC/NC : Traditional
Emergency infrastructure scaling	$15K avg.	$180K avg.	$90K avg.	12×
Performance remediation (in-system)	$80K avg.	$350K avg.	$180K avg.	4.4×
Operational downtime losses	$25K/incident	$220K/incident	$120K/incident	8.8×
System redesign / replacement	$320K avg.	$1.4M avg.	$720K avg.	4.4×
Data migration costs	$40K avg.	$280K avg.	$150K avg.	7×
Security vulnerabilities from perf. patches	$30K avg.	$190K avg.	$110K avg.	6.3×
Total per obsolescence event	$510K avg.	$2.62M avg.	$1.37M avg.	5.1×

Source: author's analysis based on empirical data from 11 Uzbekistan information systems and published industry reports, 2024

Table 3 reveals that when LC/NC-developed systems reach premature obsolescence due to algorithmic causes, the total economic impact is 5.1× greater than equivalent incidents in traditionally developed systems. This multiplier reflects the additional architectural complexity that LC/NC platforms introduce, which must be navigated before underlying algorithmic issues can be addressed.

Extrapolated to the Uzbekistan public sector - with a government IT portfolio exceeding $520M annually and LC/NC adoption growing - the expected cost of algorithm-attributable premature obsolescence over a 5-year horizon exceeds $45M. Globally, given CISQ's estimate of $2.41T in annual software quality costs and the rapid growth of LC/NC adoption, algorithm-attributable obsolescence represents a structurally growing component of this burden.

CONCLUSION

This article has investigated premature IT system obsolescence as an economic problem with a specific, identifiable structural cause: incorrect selection of algorithms and data structures at the design stage, and the systematic elimination of algorithm selection capability in LC/NC and AIDD development paradigms.

Our empirical findings establish four primary conclusions. First, algorithm complexity class is a primary determinant of IT system lifespan: O(n^2) systems average 2.8 years of useful life, compared to 12.5 years for O(log n) implementations - a 4.5×differential attributable to algorithmic choice alone. Second, the economic cost of premature obsolescence scales with algorithmic complexity: worst-case algorithm selections generate replacement costs 26× greater than optimal

implementations ($4.8M vs. $180K per incident). Third, LC/NC and AIDD platforms create a structural algorithm selection vacuum, scoring 0-4/10 on all algorithm-related capability criteria while traditional development scores 8-9/10. Fourth, when LC/NC-built systems reach premature obsolescence, remediation costs are 5.1× greater than for traditionally developed systems, compounding the initial productivity gain.

The democratization of software development through LC/NC and AIDD is a genuine achievement with substantial benefits in development speed, accessibility, and cost reduction at the application layer. However, these benefits come with a structural cost that is not adequately recognized: the systematic embedding of algorithmic technical debt into systems before development begins. The "obsolescence illusion" - systems that appear successful at launch but carry algorithmic time bombs - represents a growing and underappreciated risk in enterprise IT portfolios.

We propose three directions for future research and practice. First, the development of algorithm-aware LC/NC frameworks that expose complexity information to platform users through visual complexity indicators, performance prediction tools, and algorithm selection guidance integrated into visual development environments. Second, the establishment of algorithmic complexity standards in IT procurement, requiring vendors to document the algorithmic complexity profiles of platform components and providing performance guarantees over defined data volume ranges. Third, the integration of complexity analysis into AIDD tools, enabling AI coding assistants to consider data scale context and explicitly select appropriate algorithmic implementations based on deployment parameters.

The central challenge of the digital economy is not simply to build more software faster, but to build software that remains valuable over time. Achieving this requires restoring algorithmic decision-making as a first-class concern in all development paradigms - including those that prioritize speed and accessibility. The alternative - continuing to treat algorithm selection as an implementation detail invisible to developers and users alike - systematically produces systems designed to fail.

REFERENCES

1. IDC. Worldwide IT Spending Guide: Industry and Company Size. International Data Corporation, 2024.
2. Standish Group. CHAOS Report 2023: Decision Latency Theory. The Standish Group International, 2023.
3. CISQ. The Cost of Poor Software Quality in the US: A 2022 Report. Consortium for IT Software Quality, 2022. - 48 p.
4. Gartner. Top Strategic Technology Trends 2024: Platform Engineering and AI-Augmented Development. Gartner Research, 2024.
5. Knuth D.E. The Art of Computer Programming, Volume 3: Sorting and Searching. 2nd ed. - Addison-Wesley, 2024. - 780 p.
6. Patterson D.A., Hennessy J.L. Computer Organization and Design RISC-V Edition: The Hardware Software Interface. 2nd ed. - Morgan Kaufmann, 2021. - 696 p.
7. Fowler M. Refactoring: Improving the Design of Existing Code. 2nd ed. - Addison-Wesley, 2019. - 448 p.
8. Mo'minov B., Muhammadiyev S. Axborot tizimlarini loyihalash notatsiyalari tahlili. Digital transformation and artificial intelligence, 1(2), (2023). 79–83.
9. Muxamadiyev S. I. Raqamli iqtisodiyot sharoitida biznes jarayonlarni atomar axborot jarayonlari orqali loyihalashning yangi uslubiyoti // Innovations in Science and Technologies. — 2026. — Vol. 2, № 11. — P. 398–405.
10. Seacord R.C., Plakosh D., Lewis G.A. Modernizing Legacy Systems: Software Technologies, Engineering Processes, and Business Practices. - Addison-Wesley, 2003. - 368 p.
11. Forrester Research. The State of Low-Code/No-Code Development Platforms: 2024 Report. Forrester Research, 2024.
12. Richardson C. Microservices Patterns: With Examples in Java. - Manning Publications, 2019. -

520 p.
13. GitHub. The State of AI-Assisted Development: Copilot Impact Report 2024. GitHub Inc., 2024.
14. Sommerville I. Software Engineering. 10th ed. - Pearson, 2016. - 811 p.
15. Williams S., Waterman A., Patterson D. Roofline: An Insightful Visual Performance Model for Floating-Point Programs and Multicore Architectures. Communications of the ACM, 2009. Vol. 52(4). P. 65-76.
16. Cormen T.H., Leiserson C.E., Rivest R.L., Stein C. Introduction to Algorithms. 4th ed. - MIT Press, 2022. - 1312 p.
17. Mo‘minov B.B., Muhamadiyev S.I. Mantiqiy funksional jarayonlarni modellashtirish. “Raqamli iqtisodiyot, elektron hukumat va sun’iy intellekt uchun dasturiy vositalar, axborotlarni qayta ishlashning zamonaviy usullari” mavzusidagi respublika ilmiy-amaliy anjumani, 16.06.2023. 345-349-betlar
18. Mo’minov B.B., Muxamadiyev S.I.”Elektron interaktiv xizmatlarning jarayonlarini loyihalashtirish algoritmlari”, “Raqamli iqtisodiyot, elektron hukumat va sun’iy intellekt uchun dasturiy vositalar, axborotlarni qayta ishlashning zamonaviy usullari” Respublika ilmiy-amaliy anjumani.16-17 iyun 2023. 351-355-betlar.