Эргодизайн 2025. № 4(30). С. 486-492. ISSN 2619-1512 (print). ISSN 2658-4026 (online)
Ergodesign. 2025. no. 4(30). P. 486-492. ISSN 2619-1512 (print). ISSN 2658-4026 (online)

Научная статья
Статья в открытом доступе
УДК 331.101.1: 004.5
doi: 10.30987/2658-4026-2025-4-486-492

Эргономичность программного кода в микросервисной архитектуре

Борис Сергеевич Горячкин[1]✉, Никита Александрович Клюкин[2]
[1,2] Московский государственный технический университет им. Н.Э. Баумана, Москва, Россия
[1] bsgor@mail.ru
[2] klyukin.n21@yandex.ru

Аннотация.
В данной работе предлагается расширенный подход к оценке эргономичности программного обеспечения, объединяющий анализ классических абстракций "функция", "класс", "модуль" с особенностями микросервисных архитектур. Исследование развивает методику, основанную на правиле "7±2" и коэффициенте невязки для иерархий абстракций, адаптируя её к распределённым системам. Для микросервисов вводятся три новых метрики: коэффициент согласованности API, индекс автономности сервисов и показатель эффективности использования ресурсов, обоснованные через структурные свойства систем.

Постановка проблемы. *В условиях современного программирования с повсеместным внедрением микросервисной архитектуры, где поддержка и развитие программного обеспечения становятся приоритетными задачами, важно учитывать, что не эргономичность разрастающейся кодовой базы может привести к значительным трудовым и финансовым затратам. В связи с чем возникает необходимость оценочных инструментов эргономичности программного обеспечения в микросервисной архитектуре.*

Цель. *Разработать комплексную систему оценки эргономичности программного обеспечения, учитывающую особенности микросервисной архитектуры.*

Результаты. *Рассмотрен вопрос оценки эргономичности программного обеспечения, определены характеристики эргономичности кода. Разработаны три ключевые метрики для оценки эргономичности микросервисных архитектур.*

Практическая значимость. *Иерархический подход к проектированию и способы оценки эргономичности кода должны предоставить стандартизированный подход к оценке эргономичности программного обеспечения с микросервисной архитектурой. Это позволит упростить процесс поддержки и доработки программного обеспечения.*

Ключевые слова: микросервисная архитектура, иерархичность, оценка эргономичности, иерархичность, программное обеспечение, эргономика

Для цитирования: Горячкин Б.С., Клюкин Н.А. Эргономичность программного кода в микросервисной архитектуре // Эргодизайн. 2025. №4 (30). С. 486-492. http://dx.doi.org/10.30987/2658-4026-2025-4-486-492.

Original article
Open access article

Software Code Ergonomics in Microservice Architecture

Boris S. Goryachkin[1]✉, Nikita A. Klyukin[2]
[1,2]Bauman Moscow State Technical University, Moscow, Russia
[1] bsgor@mail.ru
[2] klyukin.n21@yandex.ru

Abstract.
This paper proposes an extended approach to assessing software ergonomics, combining the analysis of classical abstractions such as "functions", "classes", and "modules" with the peculiarities of microservice architectures. The study develops a methodology based on the "7±2" rule and inconsistency coefficient for abstraction hierarchies, adapting it to distributed systems. The work introduces three new metrics for microservices: API consistency factor, service autonomy index, and efficiency metric of resource utilization, all supported by the structural properties of systems. ***Problem Statement.***

In modern programming, with widespread adoption of microservice architecture, maintaining and evolving software become priority tasks. Poor ergonomics of growing code bases can lead to substantial labour and financial costs. Therefore, evaluation tools for software ergonomics in microservice architecture are necessary. ***The aim*** *of the study is to develop a comprehensive assessment system for software ergonomics tailored to microservice architecture. The* ***study results*** *in addressing the evaluation of software ergonomics, defining ergonomics characteristics of code, and introducing three core metrics for assessing microservice architecture ergonomics.* ***The practical significance*** *of the work lies in the fact that hierarchical design approaches and evaluation methods of code ergonomics must provide standardized means for assessing software ergonomics in microservice architecture. These tools will allow simplifying the process of maintenance and enhancement of software products.*

Keywords: microservice architecture, hierarchy, ergonomics evaluation, software, ergonomics

For citation: Goryachkin B.S., Klyukin N.A. Software Code Ergonomics in Microservice Architecture. Ergodizayn [Ergodesign]. 2025;4(30):486-492. Doi: 10.30987/10.30987/2658-4026-2025-4-486-492.

Введение

Одним из важнейших аспектов качества ПО является его эргономичность. В условиях современного программирования с повсеместным внедрением микросервисной архитектуры, где поддержка и развитие программного обеспечения становятся приоритетными задачами, важно учитывать, что не эргономичность разрастающейся кодовой базы может привести к значительным трудовым и финансовым затратам. Однако, несмотря на значимость эргономичности, на сегодняшний день отсутствуют универсальные и стандартизированные инструменты для её оценки. Вместо этого, преобладают экспертные субъективные оценки [1].

Существующие стандарты, такие как ГОСТ Р ИСО/МЭК 9126-93 "Информационная технология. Оценка программной продукции. Характеристики качества и руководство по их применению", определяют ключевые характеристики качества ПО, включая функциональность, надежность, практичность, эффективность, сопровождаемость и переносимость [2]. Однако в них не предусмотрено конкретных методик или инструментов для оценки этих характеристик, что ограничивает их практическую применимость [3] (рис. 1).

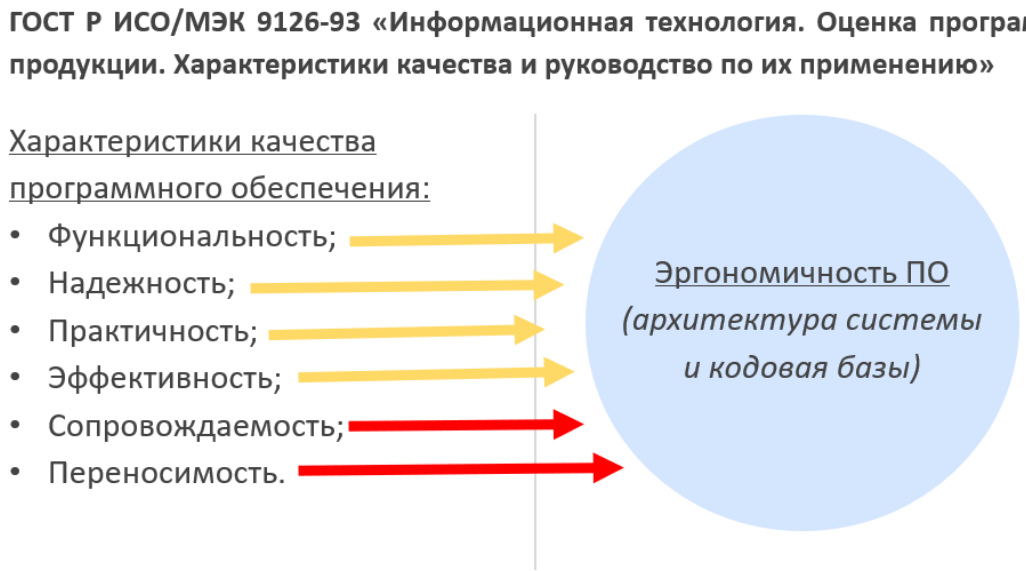


Рис. 1. Связь эргономичности ПО с ГОСТ Р ИСО/МЭК 9126-93

Fig. 1. The relation of ergonomics according to GOST R ISO/IEC 9126-93

Данное исследование направлено на решение этой проблемы путем предоставления инструментов для оценки эргономичности программного обеспечения с учётом микросервисной архитектуры.

Анализ эргономики программного обеспечения

Предложенная в данной работе модель оценки эргономичности фокусируется на оценке эргономики комплексных больших проектов, поэтому для проведения оценки абстракций возьмём 3 основные явные абстракции, названные ранее: функция, класс, модуль.

Абстракция «Функция» – это отдельный блок программы, который выполняет одно конкретное действие. Её ключевой особенностью является то, что она является самой атомарной абстракцией в следствие чего для неё крайне важным является выполнение правила «одна абстракция – одно действие».

Также для функций существует важный критерий – разрядность. Под разрядностью

функции понимается количество входящих переменных. Самыми распространенными являются нульарные, унарные, бинарные и тернарные функции. Чем меньше разрядность функции – тем лучше, поскольку позволяет снизить количество необходимых абстракций для восприятия логики кода для человека-оператора. Согласно исследованию, для эргономичного кода максимально допустимым значением разрядности является 3 [4].

Абстракция «Класс» является одной из самых часто используемых программистами абстракций, содержа в себе некоторую функциональность в виде методов и параметры в виде атрибутов. По своей сложности он является нечто средним между функцией и модулем.

По своей сути, класс является своеобразным модулем, инкапсулируя в себе определенную логику и предоставляя пользователям интерфейс для её использования. Он развивает идеи функции, добавляя в абстракцию понятие «состояние», которое достигается тем, что абстракция «класс» – также включает в себя абстракцию «объект» – конкретный экземпляр класса, обладающий определенным состоянием. Другими словами, «класс» является структурным представлением, а «объект» конкретной реализацией.

Для абстракции «метод» справедливо всё тоже самое, что и для «функции» – они являются родственными. Единственным важным отличием является то, что «метод» в своей работе может учитывать состояние объекта и его функциональность тесно связана с классом, который его определяет.

При оценке эргономичности спроектированного класса является проверка следованию им принципам объектно-ориентированного программирования.

В целом, для данной абстракции можно выделить внутреннюю иерархию, представленную деревом наследования и граф внешних взаимодействий, отображающий связь разных классов/объектов между собой, оценка которой происходить посредством оценки следования принципам ООП, изложенным выше.

Абстракция «Модуль» является самой крупный из явно поддерживаемых современными языками программирования. Она предоставляет инструмент по группировке некоторого набора функциональности с целью структуризации кодовой базы. Модуль по своей структуре состоит из двух частей: интерфейса и реализации.

Наличие интерфейса у модуля обосновывается тем, что мы хотим идентифицировать модуль как единый абстрактный объект для упрощения программистом его восприятия.

Для модулей определяют характеристики связности и сцепления [5].

Связность модуля — это мера зависимости его частей. Чем выше характеристика связности, тем ближе он к «чёрному ящику», а, значит, его проще сопровождать.

Сцепление — мера взаимозависимости модулей по данным.

Для соблюдения иерархичности необходимо исключать сцепление модулей по внешним ссылкам, поскольку это напрямую влияет на дерево взаимодействий в иерархии, неявно увеличивая количество связей, что нарушает «правило 7±2» восприятия на уровень. Следовательно, максимально допустимым значением сцепления является 4.

В рамках оценки иерархичности можно прибегнуть к оценке невязки [5].

Под невязкой подразумевают оценку отличия структуры иерархии от дерева, где 0 соответствует дереву, а 1 – полному графу, выраженную формулой:

$$Nev = \frac{2 \times (e - n + 1)}{(n - 1) \times (n - 2)},$$

Где e – количество вершин, а n – количество рёбер.

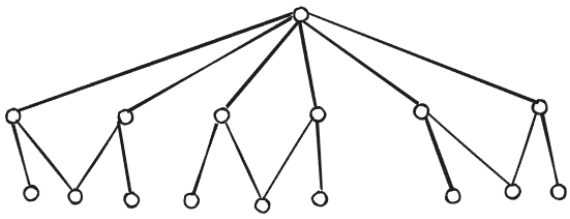
Особенностями иерархии функция является возможное наличие на 1 уровне иерархии управляющего слоя. Это вызвано тем, что функция может отвечать за выполнение одной глобальной задачи, требующее выполнения множества действий, и, в целях эргономичного размещения, допустимо выделение логического-управляющего уровня. Поэтому для расчёта коэффициента невязкости для этой иерархии абстракций следует произвести расчёт для структуры, использующей разделяемые вершины.

Для иерархии наследования классов, согласно Р. Байнедеру, крайне желательно, чтобы у каждого наследника был только один, что связано с тем, что, наследуясь от нескольких родительских классов, мы получаем из обоих свойства и методы, что сильно запутывает архитектуру.

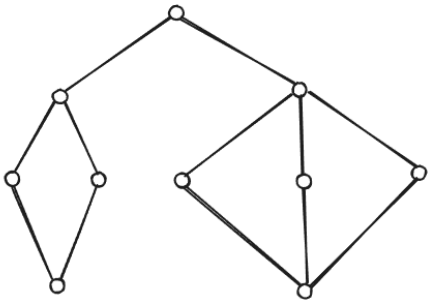
Чаще всего иерархию модулей можно условно разделить на 3 части: модули, отвечающие за отображение информации, модули бизнес-логики, модули

взаимодействия с базой данных. В такой структуре явным является факт малого количества модулей, ответственных за доступ к данным из базы данных, поскольку обычно используется единый интерфейс взаимодействия. Тоже самое справедливо и для модулей ответственных за отображение информации, поскольку они также регламентируют единый интерфейс вывода [6]. Таким образом, типовой для этой абстракции является иерархия в форме юлы – верхние и нижние уровни узкие, а средние – широкие.

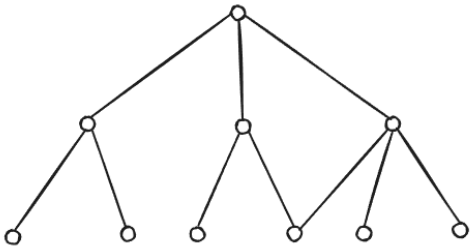
Расчёты произведём посредством искажения идеальной структуры дерева. При построении иерархии функции воспользуемся правилом: для каждой пары вершин нарушать идеальную структуру посредством добавления на уровень ниже разделяемой ими вершины. Для иерархии наследования классов: для каждой 10 вершины имеется 2 родителя. Для иерархии модуля: имеется широкий средний уровень в иерархии равный сумме вершин остальных модулей.



Иерархия функций



Иерархия модулей



Иерархия наследования классов

Рис. 2. Анализируемые типовые иерархии абстракций
Fig. 2. Typical abstraction hierarchies analyzed

Произведя соответствующие расчёты, получили для иерархии функций значение допустимой невязкости в 4%, для иерархии наследования классов – в 3%, а для иерархии модулей – в 8%.

Таблица 1.

Результаты расчётов невязкости для иерархий абстракций

Table 1.

The results of calculations of non-viscosity for hierarchies of abstractions

Функция			**Класс**			**Модуль**		
Вершин	**Рёбер**	**Невязка**	**Вершин**	**Рёбер**	**Невязка**	**Вершин**	**Рёбер**	**Невязка**
11	12	0,0444444	10	10	0,0277778	10	12	0,0833333
16	18	0,0285714	15	15	0,010989	12	14	0,0545455
21	24	0,0210526	20	21	0,0116959	13	16	0,0606061
23	26	0,017316	30	32	0,0073892	14	16	0,0384615

Результаты исследования подтверждаются практикой: так проанализировав архитектуру ПО с открытым исходным кодом видно, что все иерархии абстракции, применяемые в этом ПО приближены к идеальной структуре дерева. К таковым можно отнести иерархию

модулей Nginx (5 вершин и 4 ребра), Hadoop Cluster (9 вершин, 8 рёбер), иерархию наследования классов в Git на примере GitObject (5 вершин, 4 ребра), VTK на примере vtkObjectBase (24 вершины, 23 рёбра) [7].

Особенности оценки эргономичности микросервисов

При анализе проектов, построенных на микросервисах, было обнаружено, что коэффициент невязки для связей между сервисами существенно превышает пороговые значения, полученные ранее для иерархий модулей.

Это закономерное следствие принципиальных различий между абстракциями «модуль» и «микросервис». Если модуль представляет собой логически связанный фрагмент кода в рамках единой кодовой базы, то микросервис — это автономный компонент, обладающий независимым жизненным циклом, чётко определённым API и выделенными ресурсами. В следствие чего, использование коэффициента невязкости для данной абстракции невозможно без применения дополнительных критериев оценки. Это обуславливается рассмотрение микросервисов как отдельной единицы.

Абстракцию «микросервис» можно рассмотреть в двух плоскостях. Во-первых, её внутреннее устройство, для оценки которого подходит рассмотренный ранее подход к оценке основных абстракций через иерархичность. Во-вторых, внешнее взаимодействие, которое также можно представить в виде иерархии, однако, в связи с ранее обозначенными особенностями для этого требуется более детально рассмотреть характер этих связей.

Для этого рассмотрим основные отличительные особенности микросервисов: независимый жизненный цикл, наличие чётко определённого API, зависимость от выделенных ресурсов.

Независимый жизненный цикл микросервисов проявляется в возможности их отдельного развертывания, масштабирования и обновления без необходимости изменять другие компоненты системы. Эта характеристика напрямую влияет на эргономичность архитектуры, так как позволяет минимизировать зону воздействия при внесении изменений. Однако с точки зрения оценки эргономичности это создает дополнительные требования к согласованности версий и совместимости API, что должно учитываться при расчете метрик. Чем более автономен сервис в своем жизненном цикле, тем строже должны быть стандартизированы его интерфейсы взаимодействия с другими компонентами системы.

Чётко определённый API микросервиса представляет собой формализованный контракт, который включает не только сигнатуры методов, но и схемы данных, форматы ошибок и соглашения по версионированию. С точки зрения эргономичности это означает, что качество API документации и строгость соблюдения контрактов становятся критически важными метриками [8]. Эргономичный API микросервиса должен минимизировать когнитивную нагрузку на разработчиков, что проявляется в единообразии, предсказуемости поведения и наличии исчерпывающих примеров использования. Нарушения этих принципов существенно увеличивают коэффициент невязки на уровне межсервисного взаимодействия.

Выделенные ресурсы микросервиса подразумевают его автономность не только на логическом, но и на инфраструктурном уровне. Каждый сервис должен обладать собственными вычислительными ресурсами, хранилищами данных и механизмами обеспечения отказоустойчивости. При оценке эргономичности это приводит к необходимости учитывать новые аспекты, такие как эффективность использования ресурсов, балансировка нагрузки и стратегии резервирования. Чрезмерное дробление ресурсов между микросервисами может снизить общую эргономичность системы, так же как и их недостаточная изоляция, приводящая к конфликтам в работе компонентов [9].

Эти особенности микросервисной архитектуры требуют адаптации ранее рассмотренных метрик эргономичности. Это приводит к необходимости использования других метрик, такими как коэффициент согласованности API, индекс автономности сервисов и показатель эффективности использования ресурсов.

Коэффициент согласованности API (K_{api}) отражает степень единообразия интерфейсов взаимодействия между сервисами. Данный показатель рассчитывается как отношение количества стандартизированных параметров к общему числу параметров в API:

$$K_{api} = N_{standard}/N_{total}$$

где $N_{standard}$ — количество параметров, соответствующих принятым в системе стандартам именования, типизации и версионирования, а N_{total} — общее количество параметров во всех API системы. Высокое значение коэффициента (близкое к 1) свидетельствует о хорошей эргономичности взаимодействия между сервисами, так как разработчикам требуется запоминать меньше исключений из правил. Если на основе правила “7±2” принять, что на каждые 7 параметров допускается 1 не подходящий под принятые в системе стандарты, то получаем минимально допустимое значение коэффициента, равное в 0,85.

Индекс автономности сервисов ($I_{autonomy}$) характеризует степень независимости отдельных микросервисов. Вычисляется по формуле:

$$I_{autonomy} = 1 - (N_{dep}/N_{total\_services})$$

где N_{dep} — количество критических внешних зависимостей сервиса (другие сервисы, общие БД, глобальные очереди), а $N_{total\_services}$ — общее количество сервисов в системе. Значение индекса близкое к 1 означает полную автономность, что практически недостижимо. Заметим, что минимально используемой конфигурацией микросервисов является архитектура, состоящая из фронтенда, бэкенда и системы хранения данных, где каждый элемент высокого уровня зависит от нижнего. Таким образом, минимально допустимым значением будет 0,66.

поэтому оптимальным значением является 0,75.

Показатель эффективности использования ресурсов ($R_{efficiency}$) оценивает оптимальность распределения вычислительных мощностей между микросервисами. Рассчитывается как:

$$R_{efficiency} = \frac{\sum(U_i/C_i)}{N}$$

где U_i - фактическая утилизация ресурсов i-го сервиса, C_i - выделенная ему квота ресурсов, N - количество сервисов. Значение близкое к 1 означает идеально сбалансированное распределение ресурсов, что способствует эргономичности эксплуатации системы. Этот показатель приобретает особое значение в облачных средах, где неэффективное использование ресурсов напрямую влияет на эксплуатационные расходы. На практике принято оставлять 20-30% ресурсов для резерва.

Заключение

Таким образом, в данной работе был изучен вопрос оценки эргономичности программного кода с особенностями микросервисных архитектур. Подтверждена фундаментальная значимость правила «7±2» Джорджа Миллера для обеспечения эргономичности на всех уровнях абстракции. Работа расширяет предыдущие исследование, в котором предлагался подход к оценке эргономики программного обеспечения через анализ типовых абстракций «функция», «класс», «модуль» добавляя новые метрики для оценки микросервисов: коэффициент согласованности API, индекс автономности сервисов и показатель эффективности использования ресурсов. Также установили пороговые значения для новых метрик.

Проведённое исследование демонстрирует, что предложенный подход обеспечивает системную оценку эргономичности, учитывающую как технические аспекты проектирования, так и когнитивные нагрузки на разработчиков. Это особенно актуально в условиях роста сложности современных программных систем, где соблюдение принципов эргономичности становится критическим фактором успешной разработки и долгосрочной сопровождаемости.

СПИСОК ИСТОЧНИКОВ

1. **Pargaonkar S.** Enhancing Software Quality in Architecture Design: A Survey-Based Approach. International Journal of Scientific and Research Publications (IJSRP). 2023;13(8): 116-119. DOI 10.29322/IJSRP.13.08.2023.p14014.
2. ГОСТ Р ИСО/МЭК 25023-2021 **«Системная и программная инженерия. Требования и оценка качества систем и программного обеспечения (SQuaRE). Измерение качества системы и программной продукции».**
3. **Botchway I., Alese B.K., Agangiba W.A.** Evaluation of E-government Applications based on ISO/IEC 9126 // Anals. Computer Science. 2021;XIX(1):26-36.

REFERENCES

1. **Pargaonkar S.** Enhancing Software Quality in Architecture Design: A Survey-Based Approach. International Journal of Scientific and Research Publications (IJSRP). 2023;13(8):116-119. DOI 10.29322/IJSRP.13.08.2023.p14014.
2. **Software Engineering – Software Product Quality Requirements and Evaluation (SQuaRE). Measurement of System and Software Product Quality**. Russian Standard GOST R ISO/IEC 25023-2021. Moscow: RussianStaandardizationInstitute;2021.
3. **Botchway I., Alese B.K., Agangiba W.A.** Evaluation of E-Government Applications Based on ISO/IEC 9126. Anals. Computer Science. 2021;XIX(1):26-36.

4. **Горячкин Б.С. Бакланов Н.В.** Понятный код // E-Scio. 2023. № 2(77). С. 19-30. EDN YVYKEG.
5. **Орлов С.** Технологии разработки программного обеспечения: Учебник. СПб.: Питер, 2002. 464 с. ISBN 5-94723-820-9.
6. **Фаулер М.** Архитектура корпоративных программных приложений. М.: Вильямс, 2006. 544 с. ISBN 978-5-8459-0579-6.
7. **Chandler R., Bryant R., Bryant R.** Open source application architecture. Lulu.com , 2011. 415 p.
8. **Karabey A.I., Celik T., Can A.B., Tekinerdogan B.** Deployment and communication patterns in microservice architectures: A systematic literature review. Journal of Systems and Software. 2021;180(6):111014. DOI 10.1016/j.jss.2021.111014.
9. **Söylemez M., Tekinerdogan B., Kolukısa Tarhan A.** Challenges and Solution Directions of Microservice Architectures: A Systematic Literature Review // Applied Sciences. 2022;12(11):5507. DOI 10.3390/app12115507.

4. **Goryachkin B.S., Baklanov N.V.** Clear Code. E-Scio. 2023;2(77):19-30.
5. **Orlov S.** Software Development Technologies. Saint Petersburg: Piter; 2002. 464 p.
6. **Fowler M.** Patterns of Enterprise Application Architecture. Moscow: Williams; 2006. 544 p.
7. **Chandler R., Bryant R., Bryant R.** Open Source Application Architecture. Lulu.com; 2011. 415 p.
8. **Karabey A.I., Celik T., Can A.B., Tekinerdogan B.** Deployment and Communication Patterns in Microservice Architectures: A Systematic Literature Review. Journal of Systems and Software. 2021;180(6):111014. DOI 10.1016/j.jss.2021.111014.
9. **Söylemez M., Tekinerdogan B., Kolukısa Tarhan A.** Challenges and Solution Directions of Microservice Architectures: A Systematic Literature Review. Applied Sciences. 2022;12(11):5507. DOI 10.3390/app12115507.

Информация об авторах:
Горячкин Борис Сергеевич, кандидат технических наук, доцент; Московский государственный технический университет имени Н.Э. Баумана; г. Москва, Российская Федерация. ORCID: 0000-0002-0852-4162; РИНЦ Author ID: 683449; SPIN-код: 5465-3012; E-mail: bsgor@mail.ru
Клюкин Никита Александрович, магистрант, Московский государственный технический университет имени Н.Э. Баумана; г. Москва, Российская Федерация

Information about the authors:
Goryachkin Boris Sergeevich – Candidate of Technical Sciences, Associate Professor of Bauman Moscow State Technical University; Moscow, Russian Federation, the author's international identifiers: ORCID: 0000-0002-0852-4162; Author-ID-RSCI: 683449; SPIN-code: 5465-3012; email: bsgor@mail.ru
Klyukin Nikita Aleksandrovich– Master's Student of Bauman Moscow State Technical University; Moscow, Russian Federation

Вклад авторов: все авторы сделали эквивалентный вклад в подготовку публикации.
Contribution of the authors: the authors contributed equally to this article.
Авторы заявляют об отсутствии конфликта интересов.
The authors declare no conflicts of interests.
Статья поступила в редакцию 26.09.2025; одобрена после рецензирования 14.10.2025; принята к публикации 15.10.2025. Рецензент –Федотов С.Н., доктор психологических наук, профессор Московского университета МВД России имени В.Я. Кикотя, заместитель председателя редакционного совета журнала «Эргодизайн»
The paper was submitted for publication on the 26th of September 2025; approved after the peer review on the 14th of October 2025; accepted for publication on the 15th of October 2025. Reviewer – Fedotov S.N., Doctor of Psychology, Professor of Vladimir Kikot Moscow University of the Ministry of Internal Affairs of Russia, Deputy Chairman of the editorial board of the journal "Ergodesign".