Инженерный вестник Дона, №3 (2026)
ivdon.ru/ru/magazine/archive/n3y2026/10847

Архитектуры серверов приложений как объект системного анализа

К.И. Яскевич

Московский государственный технологический университет "СТАНКИН", г. Москва, Россия

Аннотация: Статья посвящена применению методологии системного анализа к исследованию архитектур серверов приложений. Рассматриваются принципы системного подхода применительно к программным платформам. Предложена классификация архитектур по степени распределенности, принципу компонентной организации, способу обработки запросов и применяемым архитектурным шаблонам проектирования. Проведен компонентный анализ типовой многоуровневой архитектуры с выявлением функциональных связей и механизмов взаимодействия элементов. Определены критерии оценки архитектур с позиций системного подхода, включая производительность, масштабируемость, доступность и модифицируемость. Рассмотрены методы декомпозиции и архитектурные паттерны в контексте их системных свойств, что создает основу для обоснованного выбора проектных решений в зависимости от требований конкретных задач.

Ключевые слова: системный анализ, архитектура серверов приложений, классификация архитектур, компонентный анализ, архитектурные шаблоны проектирования, декомпозиция систем, критерии оценки архитектур, монолитные архитектуры, микросервисные архитектуры, многоуровневые архитектуры.

Введение

Развитие автоматизированных и информационных систем в последние десятилетия привело к формированию сложных программных комплексов, обеспечивающих функционирование распределенных приложений. Серверы приложений стали главным элементом такой инфраструктуры, однако их проектирование и анализ долгое время опирались преимущественно на эмпирические подходы и практические рекомендации производителей. Между тем, в авторском понимании, теоретическое осмысление архитектурных решений через призму системного анализа остается недостаточно представленным в научной литературе. Существующие исследования – прежде всего, зарубежные – либо фокусируются на отдельных технологических аспектах [1], либо рассматривают архитектуру с позиций программной инженерии [2].

Исходя из обозначенного, цель статьи заключается в рассмотрении архитектур серверов приложений с позиций системного анализа и выявлении методологических подходов к их исследованию.

Методы

Системный анализ как научная дисциплина предполагает изучение объекта через выявление его структуры, связей между элементами и закономерностей функционирования [3-5]. Применительно к программным системам данный подход требует рассмотрения архитектуры не как набора технологий, а как целостной системы взаимосвязанных компонентов. В качестве основополагающих принципов здесь выступают:

- иерархичность, предполагающая многоуровневую организацию с подчинением нижних уровней верхним;
- структурность, требующая анализа устойчивых связей между элементами и их влияния на поведение системы;
- целостность, означающая наличие у системы свойств, не сводимых к свойствам отдельных компонентов;
- множественность описания, согласно которому одна архитектура может быть представлена через различные модели в зависимости от аспекта рассмотрения.

Последний принцип, в нашем представлении, особенно важен, поскольку позволяет рассматривать архитектуру одновременно с точки зрения функциональной структуры системы, то есть распределения функций между компонентами и их взаимодействия, а также распределения данных, процессов развертывания и управления ресурсами [4].

Результаты

Развивая методологические положения выше, перейдем к определению объекта исследования. Сервер приложений, в общем случае, представляет собой программную платформу, обеспечивающую выполнение бизнес-

логики и управление ресурсами для клиентских приложений. Как отмечается, в отличие от веб-сервера, который ограничивается передачей статического контента, сервер приложений предоставляет среду исполнения для программных компонентов и управляет их жизненным циклом [6]. Границы системы при этом определяются не только самой платформой, но и взаимодействием с внешними компонентами: клиентским уровнем, системами хранения данных, интеграционной шиной. Такое понимание позволяет рассматривать сервер приложений как открытую систему, обменивающуюся информацией и управляющими сигналами с окружением [2].

Отсюда возникает необходимость классификации архитектур. Единого общепринятого подхода к типологии не существует, что объясняется многомерностью объекта анализа. Исторически первой получила распространение классификация по степени распределенности вычислений, предложенная в работах 1990-х - начала 2000-х годов [7]. Монолитные архитектуры, возникшие в 1990-е годы, характеризуются размещением всех компонентов в едином развертываемом модуле (рис. 1).

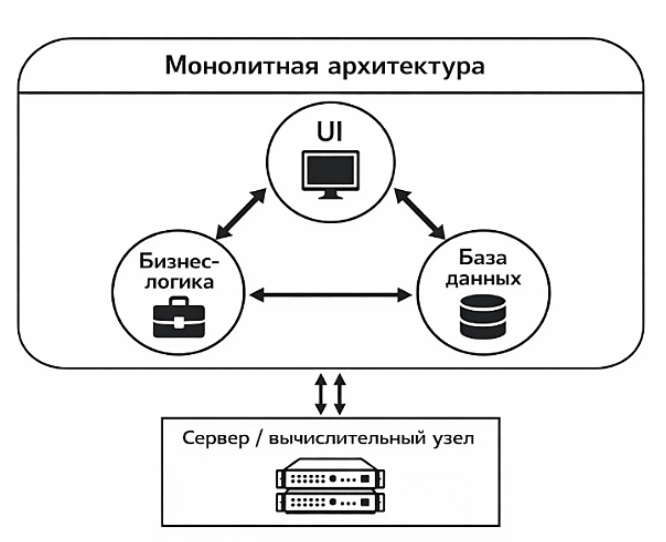

Рис. 1. – Структура монолитной архитектуры

Подобная организация, как показывает практика, упрощает разработку на начальных этапах, однако создает трудности при масштабировании и внесении изменений.

Развитием данного подхода стали многоуровневые (n-tier) архитектуры, получившие распространение в 2000-х годах. Их организация предполагает разделение на физически независимые уровни – например, в случае 3-звенной архитектуры они следующие (рис. 2):

- уровень представления, отвечающий за взаимодействие с пользователем;
- уровень бизнес-логики, реализующий правила обработки данных;
- уровень данных, обеспечивающий хранение и доступ к информации.

Рис. 2. – Трехуровневая архитектура сервера приложений

Указанная структура допускает как вертикальное масштабирование (увеличение мощности серверов), так и горизонтальное (добавление новых узлов на каждом уровне). Дальнейшее развитие привело к появлению распределенных архитектур, где компоненты размещаются на независимых узлах с возможностью динамического перераспределения нагрузки [8].

Параллельно развивалась классификация по принципу компонентной организации. Компонентно-ориентированные архитектуры основаны на повторно используемых модулях, взаимодействующих через определенные интерфейсы [9]. Здесь компонент выступает единицей развертывания и может быть заменен без перестройки всей системы. Сервис-ориентированные архитектуры расширяют данный подход, представляя функциональность через слабосвязанные сервисы, взаимодействующие посредством стандартизированных протоколов [10].

Микросервисные архитектуры, получившие распространение в середине 2010-х с развитием контейнеризации, представляют собой дальнейшую декомпозицию системы на автономные сервисы с собственными хранилищами данных, характерное схематическое изображение представлено ниже (рис. 3) [11, 12].

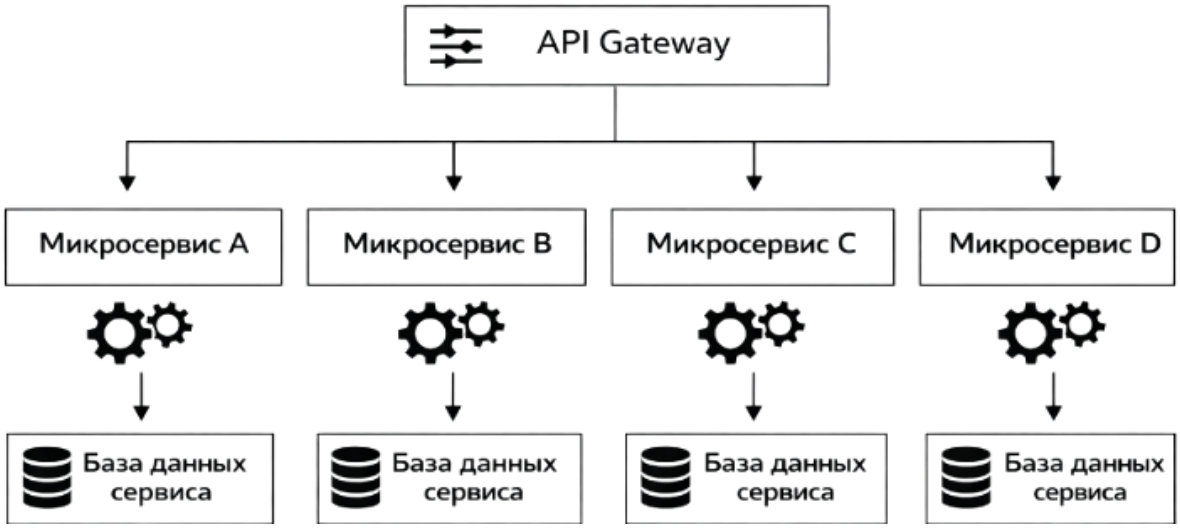

Рис. 3. – Микросервисная архитектура

Для систематизации рассмотренных подходов целесообразно представить сравнительную характеристику основных типов архитектур (табл. 1).

Таблица 1

Типология архитектур серверов приложений

Тип архитектуры	**Принцип организации**	**Степень связанности**	**Масштабируемость**
Монолитная	Единый развертываемый модуль	Высокая	Вертикальная
Многоуровневая	Разделение по слоям ответственности	Средняя	Вертикальная/горизонтальная
Сервис-ориентированная	Сервисы с общей шиной	Средняя	Горизонтальная
Микросервисная	Независимые автономные сервисы	Низкая	Горизонтальная

Помимо указанных критериев, существуют классификации по способу обработки запросов. Например, синхронные архитектуры предполагают последовательную обработку каждого запроса с ожиданием ответа перед переходом к следующей операции. Асинхронные системы используют очереди сообщений и событийно-ориентированную модель, где компоненты реагируют на события без прямого вызова [13, 14]. Данное различие влияет на производительность и отзывчивость системы: асинхронная обработка позволяет избежать блокировки потоков, однако усложняет отладку и отслеживание цепочек выполнения.

Дополнительную классификацию, которую также целесообразно отметить, образуют архитектурные паттерны, применяемые при проектировании:

- Модель-представление-контроллер (model-view-controller – MVC), разделяющий представление данных, бизнес-логику и управление взаимодействием [15];
- Многослойная архитектура (Layered architecture), организующая систему в виде иерархии уровней, каждый из которых использует сервисы нижележащего [16];
- Архитектура «Канал-Фильтр» (Pipe-and-Filter architecture) – для потоковой обработки информации, где данные последовательно проходят через цепочку преобразований [17];
- Архитектура «Черная доска» (Blackboard architecture) – для решения задач с неопределенной стратегией, где компоненты взаимодействуют через общее хранилище знаний;
- Событийно-ориентированная архитектура, основанный на асинхронной обработке событий [18].

Переходя к компонентному анализу, необходимо рассмотреть внутреннюю структуру архитектуры как системы взаимодействующих элементов. Типовая многоуровневая архитектура включает следующие подсистемы:

- уровень представления, формирующий интерфейс и управляющий взаимодействием с пользователем;
- уровень бизнес-логики, реализующий правила обработки данных, управление транзакциями и координацию компонентов;
- уровень доступа к данным, обеспечивающий абстракцию над системами хранения и управление персистентностью объектов;
- интеграционный уровень, управляющий взаимодействием с внешними системами и трансформацией данных между различными форматами.

Каждый из уровней может быть декомпозирован на более детальные компоненты. Уровень бизнес-логики включает управляющие компоненты (контроллеры), обрабатывающие входящие запросы, объекты предметной области, инкапсулирующие бизнес-правила, и сервисные компоненты, координирующие операции над группами объектов [19]. Подобная организация позволяет локализовать изменения: модификация бизнес-правил затрагивает лишь объекты предметной области, тогда как изменение порядка обработки требует корректировки контроллеров.

Связи между уровнями регулируются через определенные интерфейсы, что обеспечивает слабую связанность и возможность независимого изменения компонентов. Однако данная схема работает лишь при соблюдении принципа зависимости от абстракций, когда верхние уровни обращаются к нижним через интерфейсы, а не к конкретным реализациям [20]. Отметим, что нарушение данного принципа приводит к жесткой связанности, затрудняющей тестирование и повторное использование компонентов.

Анализ функциональных связей показывает, что взаимодействие компонентов осуществляется посредством различных механизмов. В синхронных архитектурах преобладают:

- прямые вызовы методов для локальных компонентов, обеспечивающие минимальную задержку;
- средства удаленных вызовов процедур для распределенных систем, добавляющие накладные расходы на сетевое взаимодействие;
- программные интерфейсы прикладного взаимодействия для сетевых сервисов, использующие протокол передачи гипертекста (hypertext transfer protocol – HTTP) и обеспечивающие платформенную независимость.

Асинхронные системы, в свою очередь, используют обмен сообщениями через брокеры – это повышает надежность за счет

возможности повторной обработки и хранения сообщений [21]. Выбор механизма определяется требованиями к производительности и надежности (табл. 2).

Таблица 2

Механизмы взаимодействия компонентов

Механизм	**Тип связи**	**Задержка**	**Надежность**	**Область применения**
Прямой вызов	Синхронная	Минимальная	Зависит от вызываемого	Локальные компоненты
RPC/RMI	Синхронная	Средняя	Средняя	Распределенные системы
Обмен сообщениями	Асинхронная	Высокая	Высокая	Слабосвязанные сервисы
REST API	Синхронная	Средняя	Средняя	Веб-сервисы

Рассмотренные механизмы определяют нефункциональные характеристики архитектуры, которые требуют отдельного анализа. Методы оценки архитектур с позиций системного подхода опираются на набор критериев, включающих в себя стандартные метрики:

- производительность, измеряемую через пропускную способность (количество обрабатываемых запросов в единицу времени) и время отклика системы;
- масштабируемость, определяющую способность наращивать вычислительные ресурсы без изменения архитектуры [22];
- доступность как характеристику отказоустойчивости, зависящую от наличия механизмов резервирования и восстановления после сбоев;
- модифицируемость, отражающую легкость внесения изменений в систему и напрямую связанную со степенью связанности компонентов [23].

Для количественной оценки данных характеристик применяются следующие метрики – коэффициент связанности, измеряющий зависимость между модулями, и коэффициент связности, отражающий внутреннюю

согласованность компонента [24]. Метод анализа архитектурных компромиссов (architecture tradeoff analysis method – ATAM) позволяет выявить компромиссы между различными качественными атрибутами [25]. Например, повышение производительности за счет кэширования может снизить согласованность данных, тогда как увеличение отказоустойчивости через репликацию требует дополнительных вычислительных ресурсов.

Декомпозиция архитектур представляет собой методологический прием, позволяющий упростить анализ путем разбиения системы на управляемые части. Как правило, применяются различные стратегии:

- функциональная декомпозиция, основанная на разделении обязанностей [26];
- доменная декомпозиция, выделяющая подсистемы по бизнес-областям [27];
- технологическая декомпозиция, группирующая компоненты по используемым платформам.

Выбор стратегии определяется целями анализа и спецификой системы. Функциональная декомпозиция упрощает понимание системы для технических специалистов, тогда как доменная – ближе к представлениям бизнес-аналитиков. Декомпозиция позволяет применить принцип локальности изменений, когда модификация одного компонента не требует перестройки всей архитектуры.

Выводы

Подводя итог, отметим, что применение системного анализа к архитектурам серверов приложений позволяет перейти от эмпирического выбора решений к обоснованному проектированию на основе выявленных закономерностей. Рассмотрение архитектуры как несколько-уровневой системы взаимосвязанных компонентов, анализ механизмов взаимодействия и критериев эффективности, в нашем представлении, создают

методологическую основу для оптимизации программных платформ. Классификация архитектурных подходов, анализ паттернов и методов декомпозиции дают инструментарий для обоснованного выбора решений в зависимости от требований конкретной задачи – это открывает перспективы дальнейших исследований в области формализации процессов архитектурного проектирования.

Литература

1. Fowler M. Patterns of Enterprise Application Architecture. Boston: Addison-Wesley, 2003. 533 p.
2. Bass L., Clements P., Kazman R. Software Architecture in Practice. 3rd ed. Boston: Addison-Wesley, 2012. 624 p.
3. Перегудов Ф.И., Тарасенко Ф.П. Введение в системный анализ. М.: Высшая школа, 1997. 367 с.
4. Волкова В.Н., Денисов А.А. Теория систем и системный анализ: учебник для вузов. 3-е изд. М.: Издательство Юрайт, 2024. 562 с.
5. Клиланд Д., Кинг В. Системный анализ и целевое управление. М.: Советское радио, 2004. 279 с.
6. Alur D., Crupi J., Malks D. Core J2EE Patterns: Best Practices and Design Strategies. 2nd ed. Upper Saddle River: Prentice Hall, 2003. 528 p.
7. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Reading: Addison-Wesley, 1994. 395 p.
8. Coulouris G., Dollimore J., Kindberg T., Blair G. Distributed Systems: Concepts and Design. 5th ed. Boston: Addison-Wesley, 2011. 1067 p.
9. Szyperski C. Component Software: Beyond Object-Oriented Programming. 2nd ed. Boston: Addison-Wesley, 2002. 624 p.

10. Erl T. Service-Oriented Architecture: Concepts, Technology, and Design. Upper Saddle River: Prentice Hall, 2005. 760 p.

11. Newman S. Building Microservices: Designing Fine-Grained Systems. Sebastopol: O'Reilly Media, 2015. 280 p.

12. Bernstein D. Containers and Cloud: From LXC to Docker to Kubernetes // IEEE Cloud Computing. 2014. Vol. 1, №3. pp. 81-84.

13. Hohpe G., Woolf B. Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Boston: Addison-Wesley, 2003. 683 p.

14. Michelson B.M. Event-Driven Architecture Overview // Patricia Seybold Group. 2006. №2. pp. 1-8.

15. Krasner G.E., Pope S.T. A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System // Journal of Object Oriented Programming. 1988. Vol. 1, №3. pp. 26-49.

16. Buschmann F., Meunier R., Rohnert H., Sommerlad P., Stal M. Pattern-Oriented Software Architecture: A System of Patterns. Vol. 1. Chichester: Wiley, 1996. 476 p.

17. Philipps J, Rumpe B. Refinement of Pipe-and-Filter Architectures. FM'99 - Formal Methods, Proceedings of the World Congress on Formal Methods in the Development of Computing System. LNCS 1708, pp. 96-115.

18. Chandy K.M., Schulte W.R. Event Processing: Designing IT Systems for Agile Companies. New York: McGraw-Hill, 2006. 277 p.

19. Larman C. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development. 3rd ed. Upper Saddle River: Prentice Hall, 2004. 736 p.

20. Martin R.C. Agile Software Development: Principles, Patterns, and Practices. Upper Saddle River: Prentice Hall, 2003. 529 p.

21. Tanenbaum A.S., Van Steen M. Distributed Systems: Principles and Paradigms. 2nd ed. Upper Saddle River: Prentice Hall, 2007. 686 p.

22. Abbott M.L., Fisher M.T. The Art of Scalability: Scalable Web Architecture, Processes, and Organizations for the Modern Enterprise. 2nd ed. Boston: Addison-Wesley, 2015. 464 p.

23. Clements P., Bachmann F., Bass L., Garlan D., Ivers J., Little R., Merson P., Nord R., Stafford J. Documenting Software Architectures: Views and Beyond. 2nd ed. Boston: Addison-Wesley, 2010. 592 p.

24. Stevens W.P., Myers G.J., Constantine L.L. Structured Design // IBM Systems Journal. 1974. Vol. 13, №2. P. 115-139.

25. Kazman R., Klein M., Clements P. ATAM: Method for Architecture Evaluation. Technical Report CMU/SEI-2000-TR-004. Pittsburgh: Software Engineering Institute, 2000. 83 p.

26. Parnas D.L. On the Criteria To Be Used in Decomposing Systems into Modules // Communications of the ACM. 1972. Vol. 15, №12. pp. 1053-1058.

27. Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software. Boston: Addison-Wesley, 2003. 560 p.

References

1. Fowler M. Patterns of Enterprise Application Architecture. Boston: Addison-Wesley, 2003. 533 p.

2. Bass L., Clements P., Kazman R. Software Architecture in Practice. 3rd ed. Boston: Addison-Wesley, 2012. 624 p.

3. Peregudov F.I., Tarasenko F.P. Vvedenie v sistemnyy analiz [Introduction to systems analysis]. Moskva: Vysshaya shkola, 1997. 367 p.

4. Volkova V.N., Denisov A.A. Teoriya sistem i sistemnyy analiz: uchebnik dlya vuzov [Systems theory and systems analysis: textbook for universities]. 3rd ed. Moskva: Yurayt, 2024. 562 p.

5. Kliland D., King V. Sistemnyy analiz i tselevoe upravlenie [System analysis and management by objectives]. Moskva: Sovetskoe radio, 2004. 279 p.

6. Alur D., Crupi J., Malks D. Core J2EE Patterns: Best Practices and Design Strategies. 2nd ed. Upper Saddle River: Prentice Hall, 2003. 528 p.

7. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Reading: Addison-Wesley, 1994. 395 p.

8. Coulouris G., Dollimore J., Kindberg T., Blair G. Distributed Systems: Concepts and Design. 5th ed. Boston: Addison-Wesley, 2011. 1067 p.

9. Szyperski C. Component Software: Beyond Object-Oriented Programming. 2nd ed. Boston: Addison-Wesley, 2002. 624 p.

10. Erl T. Service-Oriented Architecture: Concepts, Technology, and Design. Upper Saddle River: Prentice Hall, 2005. 760 p.

11. Newman S. Building Microservices: Designing Fine-Grained Systems. Sebastopol: O'Reilly Media, 2015. 280 p.

12. Bernstein D. Containers and Cloud: From LXC to Docker to Kubernetes. IEEE Cloud Computing, 2014, Vol. 1, No. 3, pp. 81-84.

13. Hohpe G., Woolf B. Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Boston: Addison-Wesley, 2003. 683 p.

14. Michelson B.M. Event-Driven Architecture Overview. Patricia Seybold Group, 2006, No. 2, pp. 1-8.

15. Krasner G.E., Pope S.T. A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System. Journal of Object Oriented Programming, 1988, Vol. 1, No. 3, pp. 26-49.

16. Buschmann F., Meunier R., Rohnert H., Sommerlad P., Stal M. Pattern-Oriented Software Architecture: A System of Patterns. Vol. 1. Chichester: Wiley, 1996. 476 p.

17. Philipps J, Rumpe B. Refinement of Pipe-and-Filter Architectures. FM'99 - Formal Methods, Proceedings of the World Congress on Formal Methods in the Development of Computing System. LNCS 1708, pp. 96-115.

18. Chandy K.M., Schulte W.R. Event Processing: Designing IT Systems for Agile Companies. New York: McGraw-Hill, 2006. 277 p.

19. Larman C. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development. 3rd ed. Upper Saddle River: Prentice Hall, 2004. 736 p.

20. Martin R.C. Agile Software Development: Principles, Patterns, and Practices. Upper Saddle River: Prentice Hall, 2003. 529 p.

21. Tanenbaum A.S., Van Steen M. Distributed Systems: Principles and Paradigms. 2nd ed. Upper Saddle River: Prentice Hall, 2007. 686 p.

22. Abbott M.L., Fisher M.T. The Art of Scalability: Scalable Web Architecture, Processes, and Organizations for the Modern Enterprise. 2nd ed. Boston: Addison-Wesley, 2015. 464 p.

23. Clements P., Bachmann F., Bass L., Garlan D., Ivers J., Little R., Merson P., Nord R., Stafford J. Documenting Software Architectures: Views and Beyond. 2nd ed. Boston: Addison-Wesley, 2010. 592 p.

24. Stevens W.P., Myers G.J., Constantine L.L. Structured Design. IBM Systems Journal, 1974, Vol. 13, No. 2, pp. 115-139.

25. Kazman R., Klein M., Clements P. ATAM: Method for Architecture Evaluation. Technical Report CMU/SEI-2000-TR-004. Pittsburgh: Software Engineering Institute, 2000. 83 p.

26. Parnas D.L. On the Criteria To Be Used in Decomposing Systems into Modules. Communications of the ACM, 1972, Vol. 15, No. 12, pp. 1053-1058.

27. Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software. Boston: Addison-Wesley, 2003. 560 p.

Дата поступления: 12.01.2026

Дата публикации: 3.03.2026