 © Авторы статьи / Authors of the article

СИСТЕМНЫЙ АНАЛИЗ, УПРАВЛЕНИЕ И ОБРАБОТКА ИНФОРМАЦИИ/SYSTEM ANALYSIS, MANAGEMENT AND PROCESSING OF INFORMATION

DOI: https://doi.org/10.60797/IRJ.2026.166.125 EDN: PRLXWE

ПАТТЕРН ПРОЕКТИРОВАНИЯ DRAFT SYSTEM: ЭФФЕКТИВНОЕ УПРАВЛЕНИЕ ЧЕРНОВИКАМИ В ИНФОРМАЦИОННЫХ СИСТЕМАХ

Научная статья

Андреев Р.А.[1, *]
[1] ORCID : 0009-0005-1563-6794;
[1] Хакасский государственный университет имени Н.Ф. Катанова, Абакан, Российская Федерация

* Корреспондирующий автор (grand-roman[at]yandex.ru)

Аннотация

В статье предложен и формализован паттерн Draft System для управления черновыми изменениями доменных объектов с чётким разделением подтверждённого состояния (original_entity) и временных данных (draft_data). Описана архитектурная модель, включающая абстракцию DraftSystem: доступ к черновику через raw, определение наличия правок через has_changes, а также синхронизацию и сохранение черновика посредством save_draft(). Дополнительно рассматривается компонент DraftState, реализующий жизненный цикл «инициализация–модификация–проверка–фиксация–отмена» и задающий правила переходов между стадиями. Для выявления изменений предложен механизм контроля на основе сравнения контрольных хешей состояния, позволяющий быстро определять факт модификации и снижать издержки на проверку. Показана реализация управляемого доступа к данным и централизованной проверки: DraftProperty перехватывает операции чтения/записи и обеспечивает единообразное применение черновых значений, а DraftValidator выполняет валидацию как на уровне отдельных полей, так и на уровне бизнес-правил перед публикацией изменений в подтверждённую сущность. Обсуждаются области применения и направления развития подхода, включая версионирование, распределённое хранение, поддержку аудита изменений и сценарии коллаборативного редактирования.

Ключевые слова: система управления черновиками, шаблоны проектирования, программная архитектура, управление состоянием, валидация данных.

THE DRAFT SYSTEM DESIGN PATTERN: EFFECTIVE MANAGEMENT OF DRAFTS IN INFORMATION SYSTEMS

Research article

Andreev R.A.[1, *]
[1] ORCID : 0009-0005-1563-6794;
[1] Khakassian State University named after N. F. Katanov, Abakan, Russian Federation

* Corresponding author (grand-roman[at]yandex.ru)

Abstract

The article proposes and formalises the Draft System pattern for managing draft changes to domain objects, with a clear distinction between the confirmed state (original_entity) and temporary data (draft_data). An architectural model incorporating the DraftSystem abstraction is described: access to the draft via raw, determining the presence of changes via has_changes, and synchronisation and saving of the draft via save_draft(). Additionally, the DraftState component is examined, which implements the ‘initialisation–modification–verification–commit–cancellation’ lifecycle and defines the rules for transitions between stages. To detect changes, a control mechanism based on comparing state checksums is suggested, allowing for the rapid identification of modifications and reducing verification costs. An implementation of controlled data access and centralised validation is demonstrated: DraftProperty intercepts read/write operations and ensures the consistent application of draft values, while DraftValidator performs validation both at the level of individual fields and at the level of business rules before publishing changes to the confirmed entity. The areas of application and directions for the development of the approach are discussed, including versioning, distributed storage, change audit support, and collaborative editing scenarios.

Keywords: draft system, design patterns, software architecture, state management, data validation.

Введение

Современные информационные системы всё чаще сталкиваются с проблемой потери несохранённых изменений и высокой сложностью управления промежуточными состояниями данных. В многосценарных процессах редактирования система должна уметь изолировать «незавершённые» изменения от подтверждённой версии объекта, обеспечивая согласованность, воспроизводимость и контролируемое применение правок.

Литературный обзор показывает, что в практике разработки для управления изменениями применяются как классические поведенческие паттерны, так и инфраструктурные механизмы хранения состояния. Так, паттерн Command используется для инкапсуляции операций и поддержки Undo/Redo, но он не определяет модель долговременного хранения черновых данных и правила их публикации. Memento позволяет сохранять снимки состояния, однако при масштабировании приводит к существенным накладным расходам на хранение, а также слабо отвечает на вопросы частичных (инкрементальных) изменений и доменной валидации перед фиксацией. Подход Unit

© Авторы статьи / Authors of the article

of Work и транзакционные механизмы обеспечивают согласованную фиксацию в рамках сессии, но ориентированы на короткоживущую единицу работы и не покрывают сценарии длительного ведения черновика между сессиями пользователя. State Tracking в ORM автоматизирует выявление изменений, однако привязан к жизненному циклу ORM-контекста и, как правило, не задаёт доменно-ориентированных правил безопасного редактирования, восстановления и контролируемой публикации. Следовательно, в существующем наборе подходов отсутствует целостная и доменно-ориентированная схема, которая одновременно хранит черновик независимо от транзакции ORM, допускает частичные и потенциально неконсистентные изменения, формализует проверки и обеспечивает управляемую фиксацию.

Существующие решения покрывают лишь отдельные аспекты этой задачи. Command удобен для инкапсуляции операций и поддержки отмены, но не задаёт модели хранения и жизненного цикла черновика. Memento позволяет сохранять снимки состояния, однако при практическом использовании приводит к громоздкости хранения и не отвечает на вопросы частичных изменений и валидации. Unit of Work и транзакционные механизмы ориентированы на согласованную фиксацию в рамках сессии, но не обеспечивают длительное ведение черновых данных между сессиями и не разделяют явно временное и финальное состояния. State Tracking в ORM автоматизирует отслеживание модификаций, однако остаётся привязанным к единице работы ORM и не предоставляет доменно-ориентированных правил безопасного редактирования, восстановления и публикации изменений.

В связи с этим цель работы формулируется как разработка и формализация специализированного паттерна проектирования Draft System, устраняющего указанные ограничения за счёт явного разделения подтверждённой сущности и её черновых изменений, а также формализации операций редактирования, проверки и контролируемой фиксации.

Для достижения цели решаются следующие задачи:

1. Определить ключевые требования к системам управления черновиками данных.
2. Разработать архитектурную модель Draft System, включающую основные компоненты и механизмы.
3. Оценить теоретическую и практическую значимость паттерна для различных сценариев использования.
4. Провести анализ преимуществ и возможных ограничений предложенного подхода.

Полученные результаты. В работе:

1. Сформулированы требования к системам управления черновиками данных;
2. Предложена и формализована архитектурная модель паттерна Draft System;
3. Разработан механизм отслеживания изменений на основе фиксации и сравнения контрольных хешей состояния;
4. Описаны компоненты DraftProperty и DraftValidator, обеспечивающие управляемый доступ к данным черновика и централизованную валидацию на уровне полей и бизнес-правил.

Актуальность обусловлена быстрым ростом числа приложений со сложной бизнес-логикой, когда порождается опасность потери неконсервированных данных или ошибок из-за преждевременного распространения изменений.

Научная новизна заключается в разработке паттерна Draft System как целостной доменно-ориентированной схемы управления жизненным циклом данных, которая обеспечивает хранение черновика независимо от транзакций и ORM-контекста, допускает частичные и потенциально неконсистентные промежуточные изменения и формализует операции валидации и контролируемой фиксации изменений в подтверждённую сущность [1, C. 129–130].

Оригинальность подхода состоит в объединении в одном паттерне: отдельного контейнера изменений draft_data, формализованного жизненного цикла «редактирование–проверка–публикация/отмена» и сочетания хеш-фиксации состояния с перехватом записи в свойства через DraftProperty и централизованной валидацией DraftValidator

Теоретическая значимость заключается в расширении методологии архитектурного проектирования специализированными паттернами для управления жизненным циклом данных. Практическая значимость паттерна Draft System заключается в возможности его встраивания в современные информационные системы для повышения устойчивости, надежности, управляемости и сохранности данных на всех этапах работы [2, C. 48].

Основные представления о структуризации паттерна

Паттерн Draft System описывает принцип организационного разделения рабочих состояний объектов и их подтвержденных версий. В терминах системного анализа это механизм управления модифицируемыми данными, позволяющий изолировать незавершенные изменения и впоследствии синхронизировать их с исходными сущностями по явному событию подтверждения.

Базовая структура паттерна представляет собой абстрактный класс DraftSystem, инкапсулирующий логику двухфазного взаимодействия с данными: фазу модификации и фазу персистентности. Это обеспечит транзакционность операций с сущностями и сохранит изменения изолированными до их явного подтверждения [3, C. 79].

Паттерн включает следующие ключевые элементы:

1. Оригинальная сущность (original_entity) — первичный источник данных, представляющий подтверждённое состояние доменного объекта и, в терминах DDD, выступающий в роли Aggregate Root.

2. Хранилище черновика (draft_data) — контейнер для временного сохранения изменений, реализуемый как JSON-представление набора полей доменной модели, сохраняемое в отдельном атрибуте модели базы данных. Такой подход позволяет хранить частично заполненные или несогласованные данные без нарушения целостности основной схемы, а также обеспечивает гибкость при эволюции структуры черновика.

3. Методы доступа и управления состоянием — формализованный интерфейс взаимодействия с паттерном, обеспечивающий чтение, модификацию, валидацию и фиксацию данных черновика в исходной сущности.

Структура паттерна представлена на рис. 1.

© Авторы статьи / Authors of the article

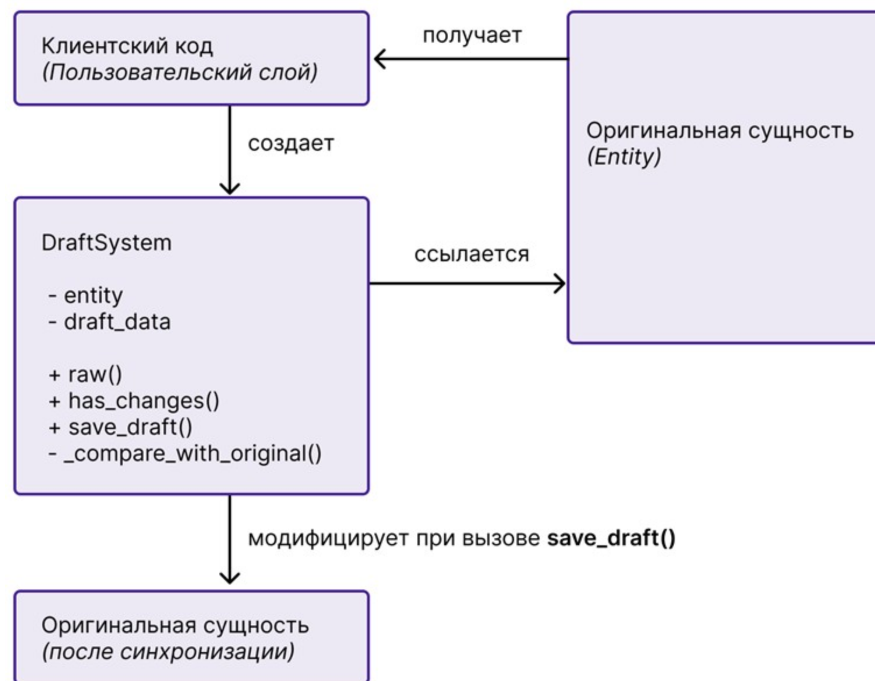

Рисунок 1 - Структура паттерна

DOI: https://doi.org/10.60797/IRJ.2026.166.125.1

Приведенная структура системы позволяет временно сохранять изменения в изолированном контейнере draft_data, предотвращая влияние на исходную сущность до момента явного сохранения. Система обеспечивает возможность отслеживания состояния модификаций через свойство has_changes, которое сравнивает черновик с оригинальной сущностью с использованием внутреннего механизма _compare_with_original(). Метод _compare_with_original() выполняет проверку по принципу хэша: при первоначальном создании черновика вычисляется контрольный хэш на основе сериализованного представления данных; затем при сравнении вычисляется хэш текущего состояния черновика тем же алгоритмом. Если хэши различаются, значит черновик был изменён. При этом доступ к данным черновика осуществляется прозрачно через свойство raw, предоставляющее прямое взаимодействие с временным хранилищем. Для синхронизации изменений предусмотрен метод save_draft(), который выполняет их контролируемую и атомарную передачу из черновика в основную сущность.

В контексте системного анализа данный архитектурный паттерн может быть классифицирован как реализация принципа разделения ответственности, где логика управления изменениями отделена от логики хранения данных. Это позволяет значительно повысить гибкость и надежность системы при работе с потенциально нестабильными состояниями объектов в процессе их модификации [5, С. 201].

Паттерн управления черновиками предусматривает эффективный механизм отслеживания изменений, который является краеугольным элементом всей архитектуры. Данный механизм обеспечивает контроль над изменениями состояния объекта и предоставляет основу для принятия решений относительно необходимости сохранения данных.

Управление в рамках паттерна реализуется через специализированный компонент DraftState, который инкапсулирует логику фиксации исходного состояния и выявления изменений. Ключевыми составляющими данного компонента являются:

1. Хранилище текущего состояния объекта.
2. Фиксатор исходного состояния (через хеш-значение).
3. Механизм сравнения текущего и исходного состояний [6, С. 121].

Механизм управления основан на вычислении хеш-значений от состояния объекта в различные моменты времени. При инициализации объекта или после подтверждения изменений фиксируется хеш текущего состояния. В последующем, сравнение этого начального хеша с хешем текущего состояния позволяет определить, были ли внесены какие-либо изменения.

На рисунке 2 представлен жизненный цикл управления состоянием.

 © Авторы статьи / Authors of the article

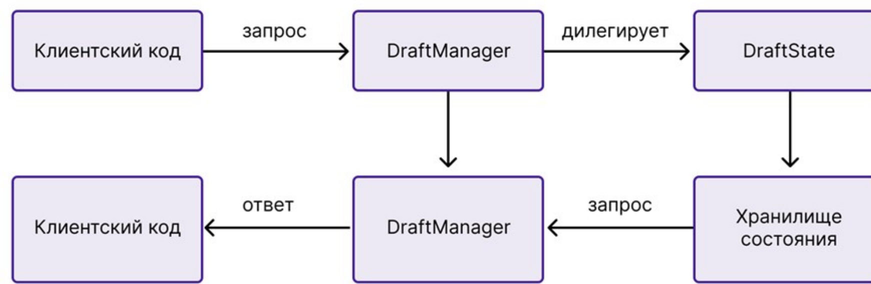

Рисунок 2 - Жизненный цикл состояния
DOI: https://doi.org/10.60797/IRJ.2026.166.125.2

Структурная схема компонента показана на рисунке 3.

Рисунок 3 - Структурная схема компонента
DOI: https://doi.org/10.60797/IRJ.2026.166.125.3

Предлагаемый механизм управления состоянием формирует надежный фундамент для построения более сложных моделей взаимодействия с данными, таких как отложенное сохранение, отмена операций или валидация изменений перед фиксацией в основном хранилище. Благодаря четкой разграниченности ответственности и контролю над жизненным циклом изменений, система приобретает значительную устойчивость к ошибкам, связанным с непреднамеренной модификацией данных [11, С. 97–120].

Реализация паттерна

Предложим далее к рассмотрению компонент DraftProperty. Он представляет собой дескриптор данных, который обеспечивает контролируемый доступ к отдельным свойствам черновика. Он инкапсулирует логику доступа к данным и их валидацию, сохраняя при этом единообразный интерфейс взаимодействия для клиентского кода [12, С. 145].

Основные функции дескриптора:

 © Авторы статьи / Authors of the article

1. Определение имени свойства для однозначной идентификации.
2. Связывание правил валидации с конкретным свойством.
3. Контроль над операциями чтения и записи данных [13, С. 213].

Дескриптор действует как промежуточный слой между клиентским кодом и внутренним представлением данных, обеспечивая согласованность и защиту состояния черновика [14, С. 104].

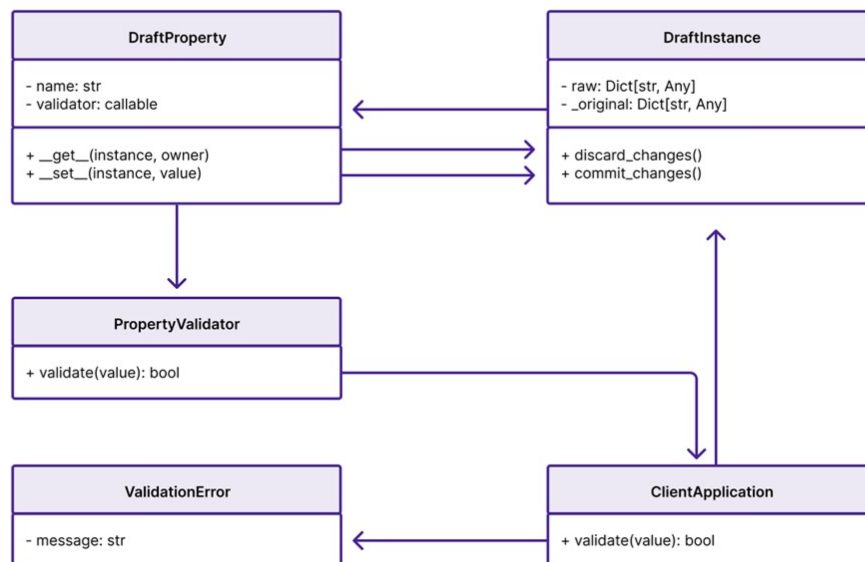

Рисунок 4 - Управление свойствами

DOI: https://doi.org/10.60797/IRJ.2026.166.125.4

Система валидации данных основана на принципе разделения ответственностей и обеспечивает проверку целостности данных как на уровне отдельных свойств, так и на уровне бизнес-правил, действующих на черновик в целом [15, С. 71–76].

Валидатор DraftValidator реализует стратегический подход к проверке корректности операций над данными черновика, что обеспечивает централизованное управление правилами целостности данных. Такой подход позволяет выполнять сложные проверки, которые охватывают несколько полей одновременно, а также предотвращать операции, способные нарушить установленную бизнес-логику [16, С. 115].

DraftValidator интегрируется с DraftProperty через механизм перехвата операций — при попытке записи DraftProperty вызывает соответствующий валидатор, передаёт ему новое значение и контекст черновика; только при успешном прохождении всех проверок DraftProperty продолжает операцию записи [17, С. 83]. На рисунке 5 показан процесс валидации: DraftProperty перехватывает запрос клиента, делегирует проверку DraftValidator и только затем, при успешной валидации, выполняет запись в черновик, тем самым исключая сохранение некорректного состояния.

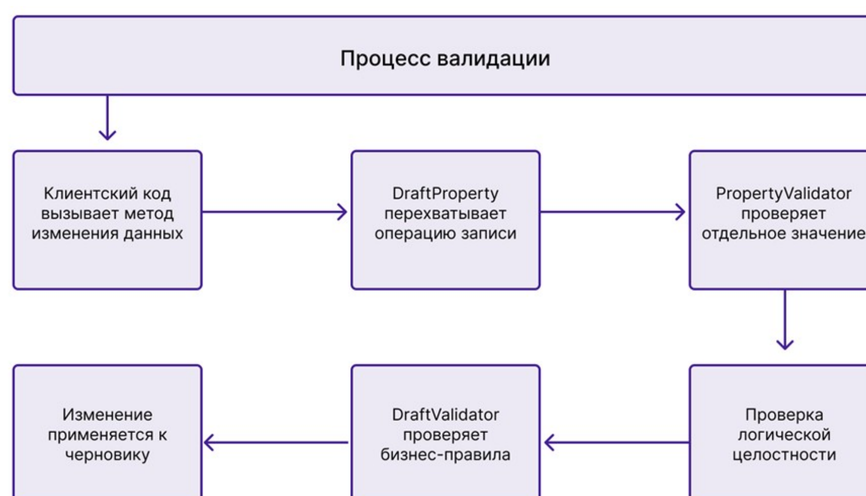

Рисунок 5 - Процесс валидации данных

DOI: https://doi.org/10.60797/IRJ.2026.166.125.5

 © Авторы статьи / Authors of the article

Компонент DraftValidator может быть специализирован для различных типов черновиков и предметных областей. Он использует полиморфизм для адаптации правил валидации к конкретным сценариям использования, сохраняя при этом общий интерфейс взаимодействия [18, С. 171].

Реализация валидации предусматривает раннее обнаружение ошибок за счет проактивного подхода, что позволяет своевременно предотвращать возможные проблемы. Процесс сопровождается информативными сообщениями об ошибках, которые полезны как для пользователей, так и для разработчиков. Также обеспечивается поддержка как синхронной, так и асинхронной проверки данных, а валидаторы могут быть комбинированы для реализации сложных бизнес-правил [19, С. 59].

Такой подход к валидации обеспечивает необходимую гибкость при расширении системы и добавлении новых типов черновиков, одновременно поддерживая строгую типизацию и согласованность данных [20, С. 141].

Заключение

Паттерн Draft System предоставляет собой эффективный механизм управления временным состоянием объектов в информационных системах. Его применение особенно целесообразно в системах с комплексной логикой изменения данных и необходимостью валидации перед окончательным сохранением.

В рамках выполненного системного анализа установлено, что данный паттерн обеспечивает необходимый уровень абстракции для разделения процессов модификации и сохранения данных, что способствует повышению надежности и гибкости архитектуры информационных систем.

Архитектурная модель, представленная в работе, демонстрирует различные аспекты реализации паттерна: от базовой структуры с компонентами DraftSystem и DraftState до специализированных механизмов управления свойствами через дескрипторы DraftProperty и валидации с помощью DraftValidator.

В современных условиях развития информационных систем, характеризующихся повышением сложности бизнес-процессов и ростом требований к надежности, паттерн Draft System представляет собой ценный инструмент в арсенале архитектора программного обеспечения.

Итоговые результаты: сформулированы требования к системам черновиков; предложена архитектурная модель паттерна Draft System; разработан механизм отслеживания изменений на основе контрольных хешей; описаны компоненты DraftProperty и DraftValidator для управляемого доступа и валидации данных черновика.

Научная новизна: предложена доменно-ориентированная схема, отделяющая подтверждённую сущность от черновых изменений и не зависящая от транзакций и ORM-контекста, с формализацией операций проверки и контролируемой фиксации.

Оригинальность: объединение в одном паттерне отдельного хранилища изменений, жизненного цикла публикации и механизма хеш-контроля изменений вместе с перехватом записи и централизованной валидацией.

Его дальнейшее совершенствование включает интеграцию с системами управления версиями, распределенными хранилищами данных и механизмами коллаборативного редактирования.

Конфликт интересов

Не указан.

Рецензия

Все статьи проходят рецензирование. Но рецензент или автор статьи предпочли не публиковать рецензию к этой статье в открытом доступе. Рецензия может быть предоставлена компетентным органам по запросу.

Conflict of Interest

None declared.

Review

All articles are peer-reviewed. But the reviewer or the author of the article chose not to publish a review of this article in the public domain. The review can be provided to the competent authorities upon request.

Список литературы на английском языке / References in English

1. Kleppmann M. Designing Data-Intensive Applications / M. Kleppmann. — Sebastopol, CA: O'Reilly Media, 2017. — 616 p.
2. Turnbull J. The Art of Monitoring / J. Turnbull. — Sebastopol, CA: O'Reilly Media, 2016. — 769 p.
3. Gamma E. Design Patterns: Elements of Reusable Object-Oriented Software / E. Gamma, R. Helm, R. Johnson [et al.]. — Boston, MA: Addison-Wesley, 1994. — 448 p.
4. Fowler M. Patterns of Enterprise Application Architecture / M. Fowler. — Boston, MA: Addison-Wesley, 2002. — 544 p.
5. Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software / E. Evans. — Boston, MA: Addison-Wesley, 2003. — 560 p.
6. Martin R.C. Clean Architecture: A Craftsman's Guide to Software Structure and Design / R.C. Martin. — Hoboken, NJ: Prentice Hall, 2017. — 432 p.
7. Buschmann F. Pattern-Oriented Software Architecture, Volume 1: A System of Patterns / F. Buschmann, R. Meunier, H. Rohnert [et al.]. — Chichester: John Wiley & Sons, 1996. — 344 p.
8. Vernon V. Implementing Domain-Driven Design / V. Vernon. — Boston, MA: Addison-Wesley, 2013. — 657 p.
9. Nystrom R. Game Programming Patterns / R. Nystrom. — Genever Benning, 2014. — 354 p.
10. Evans E. Domain-Driven Design Distilled / E. Evans, V. Vernon. — Boston, MA: Addison-Wesley, 2016. — 160 p.
11. Sussna J. Designing Delivery: Rethinking IT in the Digital Service Economy / J. Sussna. — Sebastopol, CA: O'Reilly Media, 2015. — 432 p.
12. Newman S. Building Microservices: Designing Fine-Grained Systems / S. Newman. — Sebastopol, CA: O'Reilly Media, 2015. — 624 p.

© Авторы статьи / Authors of the article

13. Richards M. Fundamentals of Software Architecture: An Engineering Approach / M. Richards, N. Ford. — Sebastopol, CA: O'Reilly Media, 2020. — 448 p.
14. Hohpe G. Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions / G. Hohpe, B. Woolf. — Boston, MA: Addison-Wesley, 2003. — 650 p.
15. Taibi D. Microservices AntiPatterns and Pitfalls / D. Taibi, V. Lenarduzzi. — Piscataway, NJ: IEEE Press, 2016. — 81 p.
16. Brown W.J. AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis / W.J. Brown, R.C. Malveau, H.W. McCormick [et al.]. — Chichester: John Wiley & Sons, 1998. — 309 p.
17. Kruchten P. The Rational Unified Process: An Introduction / P. Kruchten. — 3rd ed. — Boston, MA: Addison-Wesley, 2004. — 310 p.
18. Larman C. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design / C. Larman. — 3rd ed. — Upper Saddle River, NJ: Prentice Hall, 2004. — 736 p.
19. Metz S. Practical Object-Oriented Design: An Agile Primer Using Ruby / S. Metz. — Boston, MA: Addison-Wesley, 2018. — 288 p.
20. Feathers M. Working Effectively with Legacy Code / M. Feathers. — Upper Saddle River, NJ: Prentice Hall, 2004. — 464 p.