 © Авторы статьи / Authors of the article

ИНФОРМАТИКА И ИНФОРМАЦИОННЫЕ ПРОЦЕССЫ/INFORMATICS AND INFORMATION PROCESSES

DOI: https://doi.org/10.60797/IRJ.2026.166.58 EDN: ZOMXOK

ПРИМЕНЕНИЕ DOMAIN-DRIVEN DESIGN В ПРОМЫШЛЕННЫХ PHP-ПРОЕКТАХ: БАЛАНС МЕЖДУ ТЕОРИЕЙ И ПРАКТИКОЙ

Научная статья

Лубов В.А.[1, *]
[1] ORCID : 0009-0004-1190-5056;
[1] Московский университет им. С. Ю. Витте, Москва, Российская Федерация

* Корреспондирующий автор (et.ya.er[at]gmail.com)

Аннотация

В статье рассматриваются особенности применения подхода *Domain-Driven Design* (*DDD*) в промышленных *PHP*-проектах с акцентом на практические ограничения его использования на ранних этапах жизненного цикла программных систем. Анализ реальных сценариев разработки показывает, что прямое и полноформатное внедрение канонических рекомендаций *DDD*, включающих агрегацию сущностей, широкое использование *Value Object*, репозиториев и многоуровневых доменных слоёв, нередко приводит к избыточной архитектурной сложности, снижению темпов разработки и ухудшению сопровождаемости кода.

Научная новизна работы заключается в формализации причин неудачных внедрений *DDD* в *PHP*-системах и в предложении поэтапной модели его применения, основанной на зрелости доменной области и архитектурных потребностях проекта. Предложенный подход ориентирован на постепенное внедрение элементов *DDD* по мере роста сложности системы и позволяет сохранить доменную выразительность без преждевременного усложнения архитектуры.

Практическая значимость исследования состоит в возможности использования полученных выводов при проектировании и рефакторинге корпоративных *PHP*-приложений, а также при принятии архитектурных решений на ранних стадиях разработки.

Ключевые слова: Domain-Driven Design, PHP, архитектура программного обеспечения, доменная модель, переусложнение архитектуры, поэтапное проектирование, промышленные информационные системы.

THE APPLICATION OF DOMAIN-DRIVEN DESIGN IN INDUSTRIAL PHP PROJECTS: A BALANCE BETWEEN THEORY AND PRACTICE

Research article

Lubov V.A.[1, *]
[1] ORCID : 0009-0004-1190-5056;
[1] Moscow Witte University, Moscow, Russian Federation

* Corresponding author (et.ya.er[at]gmail.com)

Abstract

This article examines the specific aspects of applying the *Domain-Driven Design (DDD)* approach in industrial *PHP* projects, with a focus on the practical constraints of its use during the early stages of the software system lifecycle. An analysis of real-world development scenarios indicates that the direct and full-scale implementation of canonical *DDD* recommendations — including entity aggregation, extensive use of *Value Objects*, repositories and multi-level domain layers, often leads to excessive architectural complexity, a slowdown in development and a deterioration in code maintainability.

The scientific novelty of the work lies in the formalisation of the reasons behind unsuccessful *DDD* implementations in *PHP* systems and in the suggestion of a step-by-step model for its application, based on the maturity of the domain and the architectural requirements of the project. The proposed approach focuses on the gradual introduction of *DDD* elements as the system's complexity increases, and allows the domain expressiveness to be preserved without prematurely complicating the architecture.

The practical significance of this research is the possibility of using the obtained conclusions when designing and refactoring enterprise *PHP* applications, as well as when making architectural decisions in the early stages of development.

Keywords: Domain-Driven Design, PHP, software architecture, domain model, architectural over-engineering, phased design, industrial information systems.

Введение

Подход *Domain-Driven Design* (*DDD*) широко используется при проектировании сложных корпоративных информационных систем и ориентирован на построение архитектуры вокруг предметной области и бизнес-логики. В теоретических работах DDD рассматривается как универсальный метод повышения выразительности доменной модели и устойчивости архитектуры к изменениям требований.

В промышленной разработке *PHP*-систем данный подход получил распространение в связи с ростом масштабов проектов, увеличением срока их жизненного цикла и усложнением бизнес-логики [1]. Архитекторы и разработчики всё чаще стремятся применять канонические рекомендации *DDD*, опираясь на описанные в литературе шаблоны и практики.

 © Авторы статьи / Authors of the article

Практика внедрения *DDD* в *PHP*-проектах показывает наличие существенного разрыва между теоретическими рекомендациями и фактическими результатами их применения [2]. На ранних этапах жизненного цикла систем нередко осуществляется прямое и полномасштабное внедрение всех ключевых элементов *DDD*, включая агрегацию сущностей, *Value Object*, репозитории и многоуровневые архитектурные слои. В условиях отсутствия сформированной и устойчивой доменной области такой подход приводит к преждевременному усложнению архитектуры, росту объёма кода и снижению сопровождаемости программных решений [3].

Специфика *PHP* и его экосистемы усиливает данную проблему, поскольку архитектурные решения часто принимаются в условиях ограниченных сроков, необходимости быстрого вывода продукта на рынок и поддержки легаси-кода. В таких условиях избыточное применение архитектурных абстракций, характерных для полного *DDD*, не обеспечивает соразмерного выигрыша в качестве архитектуры и негативно влияет на темпы разработки.

В связи с этим актуальной задачей является исследование причин неудачных внедрений *Domain-Driven Design* в промышленных *PHP*-проектах и разработка поэтапного подхода к его применению с учётом зрелости доменной области и реальных архитектурных потребностей системы. Целью настоящей работы является формализация типовых ошибок внедрения *DDD* и разработка модели поэтапного применения его элементов, направленной на достижение баланса между теоретическими принципами и практикой промышленной разработки.

Методы и принципы исследования

Настоящее исследование выполнено в формате аналитического синтеза и основано на систематизации данных, полученных из рецензируемых научных публикаций, посвящённых практике применения *Domain-Driven Design* в промышленных программных системах, а также на обобщении архитектурного опыта проектирования и сопровождения *PHP*-приложений с различной степенью доменной и функциональной сложности. В качестве эмпирической базы исследования использовались результаты систематических обзоров литературы и опросов практикующих разработчиков, опубликованные в рецензируемых изданиях. В частности, в масштабном систематическом обзоре, выполненном группой исследователей во главе с О. Озканом и опубликованном в журнале *Journal of Systems and Software* [2], были проанализированы 36 рецензируемых исследований, посвящённых внедрению *DDD* в различных контекстах разработки. Авторы установили, что лишь около 17% из них содержат контролируемые или эмпирические оценки, тогда как большинство работ носят качественный или исследовательско-действенный характер. Другое крупное эмпирическое исследование, опубликованное в *IEEE Transactions on Software Engineering* коллективом под руководством Ч. Чжуна [4] и объединившее анализ 34 научных публикаций с опросом 63 практикующих специалистов, выявило наличие существенных вызовов, носящих общий характер для практики внедрения доменного проектирования и связанных с пониманием методологических требований *DDD* и его адаптацией к предметным областям различной сложности. Данные источники составили основу для аналитического обобщения, выполненного в настоящей работе.

Методологически исследование опирается на принципы аналитического синтеза: систематическое извлечение, сопоставление и обобщение архитектурных наблюдений из опубликованных источников с целью формирования целостной интерпретации выявленных закономерностей. В отличие от эмпирических исследований, основанных на контролируемых экспериментах, аналитический синтез направлен на выявление устойчивых паттернов и формулирование концептуальных моделей на основе имеющейся доказательной базы [3]. Такой подход является общепринятым в области архитектуры программного обеспечения при анализе проектных решений и их последствий в контексте промышленной разработки.

В рамках исследования рассматривались архитектурные решения, применяемые при построении доменной модели *PHP*-приложений, включая способы использования основных элементов *Domain-Driven Design*: сущностей (*Entity*), объектов-значений (*Value Object*), агрегатов (*Aggregate*), репозиториев (*Repository*) и доменных сервисов (*Domain Service*) [5]. Анализ фокусировался на том, каким образом данные элементы интегрируются в архитектуру системы, какую практическую роль они выполняют и как влияют на сложность, сопровождаемость и эволюцию кода на различных этапах жизненного цикла проекта [6].

Для формализации результатов использовался метод выявления архитектурных симптомов, позволяющий описывать повторяющиеся проектные решения, формально соответствующие рекомендациям *Domain-Driven Design*, но не обеспечивающие практической архитектурной ценности в условиях недостаточно сформированной предметной области или ранней стадии развития системы. Под архитектурным симптомом в рамках данной работы понимается устойчивый шаблон использования элементов *DDD*, приводящий к росту архитектурной сложности без эквивалентного улучшения качества доменной модели [7]. Идентификация симптомов проводилась на основе критериев, включающих: наличие доменных инвариантов, обосновывающих выделение агрегатов; содержательность бизнес-логики в объектах-значениях; степень дублирования функциональности между репозиториями и *ORM*; соответствие уровня архитектурных абстракций фактической сложности предметной области. Данные критерии были сформулированы на основе рекомендаций, описанных в работах В. Хононова [8], В. Вернона [1] и др., и адаптированы к контексту промышленной *PHP*-разработки.

Таблица 1 - Архитектурные симптомы избыточного применения Domain-Driven Design в PHP-проектах

DOI: https://doi.org/10.60797/IRJ.2026.166.58.1

Элемент DDD	Архитектурный симптом	Практическое последствие
Aggregate	Выделение агрегатов при отсутствии инвариантов и	Избыточная структурная сложность доменной модели

 © Авторы статьи / Authors of the article

Элемент DDD	Архитектурный симптом	Практическое последствие
	согласованного жизненного цикла	
Value Object	Формальное использование объектов-значений без доменной логики	Увеличение количества классов и снижение читаемости
Repository	Репозитории, дублирующие интерфейсы ORM без дополнительной семантики	Лишний уровень абстракции и усложнение трассировки данных
Domain Service	Доменные сервисы с процедурной логикой	Формирование анемичной доменной модели
Архитектурные слои	Многоуровневая архитектура без чёткого распределения ответственности	Рост связности и сложности сопровождения

Для обобщения наблюдений применялся сравнительный анализ архитектурных подходов, при котором сопоставлялись сценарии раннего догматического внедрения полного набора *DDD*-паттернов и поэтапного применения элементов доменного проектирования по мере роста сложности предметной области. Сравнение проводилось на уровне архитектурных характеристик системы, включая прозрачность доменной логики, гибкость внесения изменений и уровень архитектурного шума. В качестве индикаторов архитектурной сложности использовались качественные показатели: количество архитектурных слоёв и абстракций, степень дублирования функциональности между слоями, наличие формальных доменных компонентов без содержательной бизнес-логики, а также трудоёмкость внесения типовых изменений в систему. Данный подход согласуется с методологией анализа архитектурных решений, применяемой в работах [2], [4], [9].

 © Авторы статьи / Authors of the article

Рисунок 1 - Поэтапное применение Domain-Driven Design в зависимости от зрелости проекта
DOI: https://doi.org/10.60797/IRJ.2026.166.58.2

Используемая методология позволяет рассматривать *Domain-Driven Design* как адаптивный архитектурный инструмент, а не как фиксированный набор обязательных паттернов, и обеспечивает практико-ориентированную интерпретацию результатов исследования.

Основные результаты

В результате проведённого исследования были выявлены устойчивые закономерности применения *Domain-Driven Design* в промышленных *PHP*-проектах, связанные с несоответствием между стадией зрелости системы и используемым набором доменных абстракций. Полученные результаты подтверждают, что архитектурные проблемы, возникающие при внедрении *DDD*, в большинстве случаев обусловлены не сложностью предметной области, а преждевременным и избыточным применением его канонических элементов [9]. Данный вывод согласуется с результатами систематического обзора, выполненного О. Озканом и др. [2], в котором отмечается, что успешность внедрения *DDD* существенно зависит от экспертизы участников проекта и глубины понимания предметной области, а также с данными эмпирического исследования Ч. Чжуна и др. [4], фиксирующими общие для индустрии трудности в понимании методологических требований *DDD* при его практическом применении в промышленных проектах.

Установлено, что на ранних этапах жизненного цикла *PHP*-проектов формальное внедрение агрегатов, объектов-значений и репозиториев при отсутствии чётко определённых доменных инвариантов приводит к росту архитектурной сложности без сопоставимого увеличения выразительности доменной модели. В таких системах наблюдается увеличение количества классов и слоёв, усложнение навигации по коду и размывание ответственности между компонентами, что негативно влияет на сопровождаемость и скорость эволюции приложения.

 © Авторы статьи / Authors of the article

Результаты анализа показали, что характер выявленных архитектурных проблем тесно связан со стадией зрелости проекта. На ранней стадии преобладают ошибки, связанные с преждевременным введением ключевых элементов *DDD*, тогда как на стадии роста основным источником архитектурного шума становится неконтролируемое расширение доменной модели без пересмотра ранее принятых решений. В зрелых системах подобные проблемы, как правило, устраняются за счёт упрощения архитектуры или отказа от части доменных абстракций.

Таблица 2 - Результаты анализа применения Domain-Driven Design на различных стадиях зрелости PHP-проекта

DOI: https://doi.org/10.60797/IRJ.2026.166.58.3

Стадия зрелости проекта	Характер применения DDD	Основной архитектурный результат
Ранняя стадия	Полномасштабное внедрение агрегатов, *Value Object* и репозиториев	Преждевременное усложнение архитектуры
Стадия роста	Частичное упрощение или выборочное использование элементов *DDD*	Снижение связности при сохранении архитектурного шума
Зрелая стадия	Осознанное и избирательное применение доменных абстракций	Повышение устойчивости и сопровождаемости системы

Дополнительно установлено, что проекты, в которых элементы *Domain-Driven Design* внедрялись поэтапно, демонстрируют более предсказуемую архитектурную эволюцию. В таких системах доменные абстракции вводятся по мере необходимости и сохраняют прямую связь с реальными потребностями предметной области, что снижает риск накопления избыточных архитектурных решений. К аналогичным выводам пришли авторы работы О. Озкан и др. [9], проведённой в рамках методологии исследования действием (*action research*): применение *DDD* при рефакторинге промышленной системы продемонстрировало повышение сопровождаемости кода именно при условии поэтапного и осознанного введения доменных абстракций. Поэтапный характер внедрения *DDD* позволяет рассматривать доменное проектирование как инструмент адаптации архитектуры, а не как фиксированный шаблон, применяемый независимо от контекста.

Концептуальная модель зависимости архитектурной сложности от стадии зрелости проекта и подхода к внедрению *DDD* (рисунок 2) построена методом экспертного синтеза на основе качественных закономерностей, выявленных в ходе анализа литературных источников [2], [4], [9] и обобщения архитектурного опыта. Ось абсцисс отражает стадию зрелости проекта (ранняя, рост, зрелая), ось ординат — относительный уровень архитектурной сложности. Две кривые представляют альтернативные стратегии: догматическое внедрение полного набора *DDD*-паттернов с начала разработки и поэтапное введение доменных абстракций по мере роста сложности предметной области. Область между кривыми обозначена как зона преждевременного переусложнения и отражает избыточную архитектурную сложность, возникающую при догматическом подходе на ранних и средних стадиях проекта. Модель носит качественный характер и предназначена для визуализации выявленной закономерности, а не для количественного прогнозирования.

 © Авторы статьи / Authors of the article

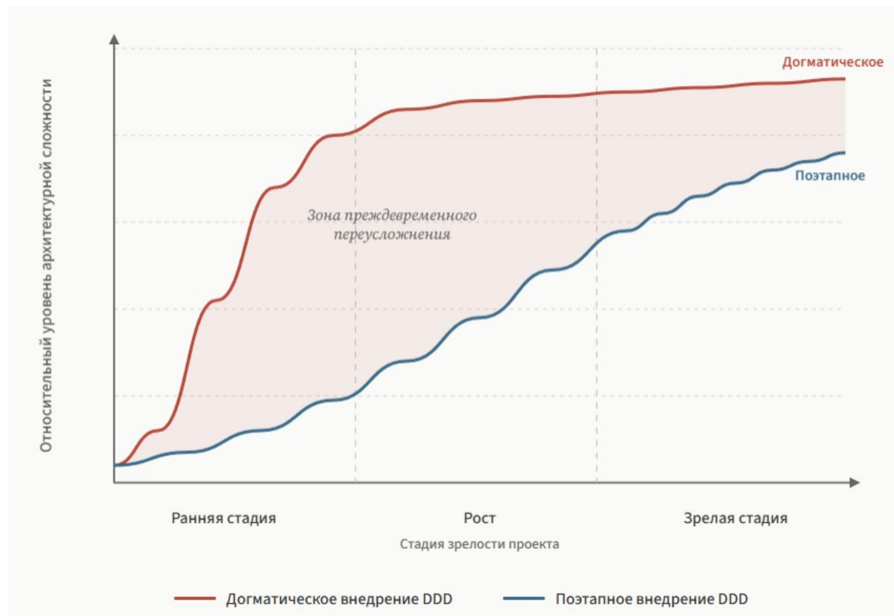

Рисунок 2 - Зависимость архитектурной сложности PHP-проекта от стадии зрелости и глубины внедрения DDD
DOI: https://doi.org/10.60797/IRJ.2026.166.58.4

Таким образом, результаты исследования подтверждают, что поэтапное применение *Domain-Driven Design*, согласованное со стадией зрелости *PHP*-проекта, снижает риск архитектурных ошибок и способствует формированию более устойчивых и сопровождаемых программных систем. Полученные выводы логически согласуются с концептуальной моделью поэтапного внедрения *DDD*, представленной на рисунке 1, и дополняют её аналитической интерпретацией.

Обсуждение результатов

Полученные результаты подтверждают, что основная причина неудачных внедрений *Domain-Driven Design* в промышленных *PHP*-проектах заключается не в ограничениях самого подхода, а в его догматическом применении без учёта стадии зрелости системы и степени сформированности предметной области. Практика показывает, что попытка реализовать полный набор архитектурных элементов *DDD* на ранних этапах разработки приводит к возникновению избыточной сложности, не компенсируемой повышением выразительности доменной модели. Данное наблюдение согласуется с выводами систематического обзора [2], в котором отмечается необходимость эмпирически обоснованного подхода к внедрению *DDD*, а также с результатами опроса практикующих специалистов [4], зафиксировавшего высокий порог входа и дефицит методического сопровождения как ключевые препятствия успешного применения доменного проектирования.

Сравнение выявленных закономерностей с каноническими рекомендациями *Domain-Driven Design* показывает, что многие из описываемых в литературе паттернов предполагают наличие устойчивой и хорошо формализованной предметной области [8], [10]. В условиях промышленной *PHP*-разработки такие предпосылки часто отсутствуют на начальных этапах проекта, что делает прямое следование канонической модели *DDD* архитектурно неоправданным. В результате доменные абстракции используются формально и утрачивают свою исходную роль как инструменты выражения бизнес-логики. Как показано в исследовании [9], успешный рефакторинг промышленной системы с применением *DDD* требует не только технической компетенции, но и глубокого вовлечения доменных экспертов, что на ранних стадиях разработки зачастую недостижимо.

Особую роль в выявленных архитектурных проблемах играет специфика *PHP* и его экосистемы. Для *PHP*-проектов характерны быстрые итерации разработки, высокая доля легаси-кода и необходимость оперативного реагирования на изменения требований [11]. В таких условиях избыточное количество слоёв и абстракций усложняет сопровождение системы и снижает адаптивность архитектуры. Это объясняет, почему преждевременное внедрение агрегатов, репозиториев и объектов-значений в *PHP*-системах оказывает более выраженное негативное влияние, чем в проектах, изначально ориентированных на строгие архитектурные ограничения.

Предложенная в работе поэтапная модель применения *Domain-Driven Design* позволяет интерпретировать *DDD* не как жёсткий архитектурный шаблон, а как набор проектных инструментов, применяемых избирательно и в зависимости от контекста. Такая интерпретация согласуется с результатами исследования и позволяет снизить риск архитектурного переусложнения, сохраняя при этом преимущества доменного подхода на тех стадиях жизненного цикла системы, где он действительно необходим. Сравнительная динамика архитектурной сложности при различных подходах к внедрению *DDD*, представленная на рисунке 2, наглядно демонстрирует преимущества поэтапного применения доменных абстракций.

Следует отметить, что результаты исследования ориентированы на анализ промышленной *PHP*-разработки и не претендуют на универсальность для всех технологических платформ. В системах с иными архитектурными предпосылками и процессами разработки характер выявленных закономерностей может отличаться. Тем не менее предложенная модель поэтапного применения *Domain-Driven Design* может рассматриваться как практико-

 © Авторы статьи / Authors of the article

ориентированная основа для принятия архитектурных решений в условиях ограниченных ресурсов и высокой изменчивости требований.

Заключение

В рамках настоящего исследования были рассмотрены особенности применения *Domain-Driven Design* в промышленных *PHP*-проектах с точки зрения соответствия канонических архитектурных рекомендаций практическим условиям разработки. Проведённый анализ показал, что основные архитектурные проблемы при внедрении *DDD* возникают не вследствие ограничений самого подхода, а в результате его преждевременного и догматического применения на ранних этапах жизненного цикла программных систем.

Установлено, что использование полного набора элементов *Domain-Driven Design* при отсутствии сформированной предметной области приводит к избыточной архитектурной сложности, росту объёма кода и снижению сопровождаемости *PHP*-приложений. В таких условиях доменные абстракции утрачивают свою исходную функциональную роль и используются формально, что препятствует дальнейшей эволюции архитектуры и формированию выразительной доменной модели.

Научная новизна работы заключается в формализации типовых архитектурных ошибок внедрения *Domain-Driven Design* в *PHP*-системах и в предложении поэтапной модели его применения, ориентированной на стадию зрелости проекта и реальную сложность предметной области. Представленный подход позволяет рассматривать *DDD* как адаптивный набор архитектурных инструментов, а не как универсальный шаблон, обязательный для применения на всех этапах разработки [8].

Практическая значимость исследования состоит в возможности использования полученных выводов при проектировании и рефакторинге промышленных *PHP*-приложений, а также при принятии архитектурных решений на ранних стадиях жизненного цикла проектов. Поэтапное внедрение элементов *Domain-Driven Design* снижает риск архитектурного переусложнения и способствует формированию более устойчивых и сопровождаемых программных систем.

Полученные результаты могут быть использованы в дальнейших исследованиях, направленных на эмпирическую оценку архитектурных подходов в других технологических экосистемах, а также на разработку методик автоматизированного анализа архитектурной сложности и зрелости доменных моделей в промышленных программных системах.

Конфликт интересов

Не указан.

Conflict of Interest

None declared.

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.

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 / References

1. Vernon V. Strategic Monoliths and Microservices: Driving Innovation Using Purposeful Architecture / V. Vernon, T. Jaskul. — Addison-Wesley, 2022. — 352 p.
2. Ozkan O. Domain-Driven Design in software development: A systematic literature review on implementation, challenges, and effectiveness / O. Ozkan, O. Babur, M. van den Brand // Journal of Systems and Software. — 2025. — Vol. 230. — Art. 112537. — DOI: 10.1016/j.jss.2025.112537.
3. Ford N. Software Architecture: The Hard Parts: Modern Trade-Off Analyses for Distributed Architectures / N. Ford, M. Richards, P. Sadalage [et al.]. — O'Reilly Media, 2021.
4. Buenosvinos C. Domain-Driven Design in PHP / C. Buenosvinos, C. Soronellas, K. Akbary. — Birmingham: Packt Publishing, 2017. — 394 p.
5. Zandstra M. PHP 8 Objects, Patterns, and Practice: Mastering OO Enhancements, Design Patterns, and Essential Development Tools / M. Zandstra. — New York: Apress, 2021. — DOI: 10.1007/978-1-4842-6791-2.
6. Ford N. Building Evolutionary Architectures: Automated Software Governance / N. Ford, R. Parsons, P. Kua [et al.]. — O'Reilly Media, 2022.
7. Ozkan O. Refactoring with domain-driven design in an industrial context: An action research report / O. Ozkan, O. Babur, M. van den Brand [et al.] // Empirical Software Engineering. — 2023. — Vol. 28. — Art. 94. — DOI: 10.1007/s10664-023-10310-1.
8. Levezinho M. Domain-Driven Design Representation of Monolith Candidate Decompositions / M. Levezinho, S. Kapferer, O. Zimmermann [et al.] // Enterprise Design, Operations, and Computing. EDOC 2024. Lecture Notes in Computer Science. — Springer, 2025. — Vol. 15409. — P. 182–200. — DOI: 10.1007/978-3-031-78338-8_10.
9. Бурдуков В.П. Domain-Driven Design: преимущества и недостатки методологии, управляемой предметной областью / В.П. Бурдуков // Научный аспект. — 2024. — №7.
10. Rio A. PHP code smells in web apps: Evolution, survival and anomalies / A. Rio, F. Brito e Abreu // Journal of Systems and Software. — 2023. — Vol. 200. — Art. 111644. — DOI: 10.1016/j.jss.2023.111644.
11. Хононов В. Изучаем DDD — предметно-ориентированное проектирование / В. Хононов. — Санкт-Петербург: БХВ-Петербург, 2024. — 320 с.

 © Авторы статьи / Authors of the article

Список литературы на английском языке / References in English

1. Vernon V. Strategic Monoliths and Microservices: Driving Innovation Using Purposeful Architecture / V. Vernon, T. Jaskul. — Addison-Wesley, 2022. — 352 p.
2. Ozkan O. Domain-Driven Design in software development: A systematic literature review on implementation, challenges, and effectiveness / O. Ozkan, O. Babur, M. van den Brand // Journal of Systems and Software. — 2025. — Vol. 230. — Art. 112537. — DOI: 10.1016/j.jss.2025.112537.
3. Ford N. Software Architecture: The Hard Parts: Modern Trade-Off Analyses for Distributed Architectures / N. Ford, M. Richards, P. Sadalage [et al.]. — O'Reilly Media, 2021.
4. Buenosvinos C. Domain-Driven Design in PHP / C. Buenosvinos, C. Soronellas, K. Akbary. — Birmingham: Packt Publishing, 2017. — 394 p.
5. Zandstra M. PHP 8 Objects, Patterns, and Practice: Mastering OO Enhancements, Design Patterns, and Essential Development Tools / M. Zandstra. — New York: Apress, 2021. — DOI: 10.1007/978-1-4842-6791-2.
6. Ford N. Building Evolutionary Architectures: Automated Software Governance / N. Ford, R. Parsons, P. Kua [et al.]. — O'Reilly Media, 2022.
7. Ozkan O. Refactoring with domain-driven design in an industrial context: An action research report / O. Ozkan, O. Babur, M. van den Brand [et al.] // Empirical Software Engineering. — 2023. — Vol. 28. — Art. 94. — DOI: 10.1007/s10664-023-10310-1.
8. Levezinho M. Domain-Driven Design Representation of Monolith Candidate Decompositions / M. Levezinho, S. Kapferer, O. Zimmermann [et al.] // Enterprise Design, Operations, and Computing. EDOC 2024. Lecture Notes in Computer Science. — Springer, 2025. — Vol. 15409. — P. 182–200. — DOI: 10.1007/978-3-031-78338-8_10.
9. Burdukov V.P. Domain-Driven Design: preimushchestva i nedostatki metodologii, upravlyaemoi predmetnoi oblastyu [Domain-Driven Design: pros and cons of a domain-driven methodology] / V.P. Burdukov // Nauchnii aspekt [Scientific Aspect]. — 2024. — №7. [in Russian]
10. Rio A. PHP code smells in web apps: Evolution, survival and anomalies / A. Rio, F. Brito e Abreu // Journal of Systems and Software. — 2023. — Vol. 200. — Art. 111644. — DOI: 10.1016/j.jss.2023.111644.
11. Khononov V. Izuchaem DDD — predmetno-orientirovannoe proektirovanie [Exploring DDD – Domain-Driven Design] / V. Khononov. — St.Petersburg: BKhV-Petersburg, 2024. — 320 p. [in Russian]