УДК 004.415.2

КОНЦЕПЦИЯ ПРЕДМЕТНО-ОРИЕНТИРОВАННОГО ПРОЕКТИРОВАНИЯ: ПОЛОЖИТЕЛЬНЫЕ И ОТРИЦАТЕЛЬНЫЕ АСПЕКТЫ ПРАКТИЧЕСКОГО ПРИМЕНЕНИЯ

А.А. Файтельсон
студент 2 курса по направлению подготовки «Математическое обеспечение и администрирование информационных систем»
Курский государственный университет
e-mail: z0tedd@gmail.com

Научный руководитель:
А.В. Кривонос
Кандидат технических наук, доцент кафедры программного обеспечения и администрирования информационных систем
Курский государственный университет
e-mail: krivonos_av@kursksu.ru

Статья посвящена исследованию подхода проектирования, управляемого предметной областью (Domain-Driven Design, DDD), который становится все более актуальным в разработке программного обеспечения. Основная цель DDD – устранение разрыва между бизнесом и разработкой, позволяющее командам создавать более точные и поддерживаемые системы, глубоко интегрированные с бизнес-процессами.

Ключевые слова: *Domain-Driven Design, DDD, разработка программного обеспечения, архитектура.*

Введение

Подход Domain-Driven Design (DDD) был предложен Эриком Эвансом [Эванс 2017] в начале 2000-х годов как ответ на растущие сложности в разработке корпоративного ПО. Основная идея DDD заключается в том, чтобы максимально сосредоточиться на моделировании предметной области – совокупности бизнес-процессов и правил, характерных для конкретной компании или организации. DDD стремится устранить разрыв между бизнесом и разработкой, предлагая общий язык и методики, которые делают модели системы понятными для всех участников процесса от разработчиков до бизнес-аналитиков и заказчиков.

DDD становится незаменимым при работе над системами со сложной структурой, где важна высокая точность в передаче бизнес-логики от заказчика к разработчику. Использование DDD позволяет разработчикам лучше понимать суть бизнес-процессов, выделяя ключевые сущности и связи, которые играют важную роль для компании. Таким образом, разработка ПО идет не в отрыве от бизнес-логики, а в тесной

интеграции с ней, что помогает создавать ПО, действительно решающее задачи бизнеса.

Преимущества DDD для разработки программного обеспечения

Одним из главных преимуществ DDD является возможность наладить взаимодействие между техническими специалистами и стейкхолдерами. Используя DDD, компании формируют «общее видение» – когда разработчики, бизнес-аналитики и представители заказчика начинают работать как единая команда. Это общее понимание достигается благодаря созданию "общего языка" (Ubiquitous Language), который используется в повседневном общении и отражается в коде, моделях и документации. Общий язык позволяет снизить количество ошибок, вызванных недопониманием требований, и сделать требования к ПО более однозначными и структурированными.

DDD помогает значительно улучшить качество кода и архитектуры [Вернон 2023]. Код структурируется так, что его легче поддерживать, он становится более устойчивым к изменениям. Благодаря выделению основных сущностей и паттернов, DDD-система остается логичной и предсказуемой. К тому же этот подход снижает «долг по документации», так как сама структура кода и модели служат наглядной документацией системы.

Основные концепции и элементы DDD

Каждая программа связана с определенной деятельностью или интересами пользователя. Предметная область, к которой пользователь применяет программу, называется ее «доменом» [Эванс 2017: 28]. Некоторые домены трудно осязаемы физически – например, областью применения банковского API являются операции над денежными активами. Но существуют и обратные случаи, когда предметная область крепко привязана к физическому миру: к примеру, домен сайта покупки билетов ОАО «РЖД» включает в себя людей, которые садятся в реально существующие поезда. В основе DDD лежит доменная модель, которая описывает предметную область и отражает реальные бизнес-процессы. Доменные модели создаются в тесном сотрудничестве с бизнесом, чтобы максимально точно передавать смысл и цели, которые должны решаться в приложении. Такие модели становятся своего рода «картой» для разработки, обеспечивая разработчикам четкое представление о структуре и задачах системы.

DDD выделяет три основных элемента модели, каждый из которых выполняет свою роль [Дандан 2022]:

1. Сущности (Entities) – объекты с уникальной идентичностью, которые изменяются во времени, но остаются узнаваемыми (например, клиент в CRM-системе).

2. Объекты-значения (Value Objects) – объекты, не имеющие уникальной идентичности, но содержащие данные (например, адрес). Они неизменяемы и применяются для группировки значений.

3. Агрегаты (Aggregates) – совокупности сущностей и объектов-значений, объединенные вокруг главной сущности, называемой корнем агрегата (Aggregate Root). Агрегаты помогают контролировать целостность данных и обеспечивать атомарные операции.

Доменные события фиксируют важные моменты в жизненном цикле объекта или процесса. Это не просто данные, но и сигнал для системы о том, что произошло значимое изменение. Также в DDD активно используется паттерн проектирования «репозиторий». Репозиторий представляет все объекты определенного типа в виде абстрактного набора. Он действует как коллекции, за исключением возможности более сложных запросов. Репозитории обеспечивают доступ к данным агрегатов и позволяют реализовать принципы DDD, сохраняя доменную модель независимой от деталей хранения данных.

При интеграции с внешними системами DDD предлагает использовать так называемые антикоррупционные слои, чтобы избежать прямого воздействия внешней логики на внутреннюю модель предметной области. Антикоррупционные слои – изолированные интерфейсы, предоставляющие функциональность в контексте их предметной области. Данные слои общаются с другой системой через существующие методы, которые или не требуют модификаций другой системы, или требуют самый минимум внесения изменений. Это особенно важно для защиты предметной области от нежелательных влияний, позволяя системе сохранять устойчивость и единообразие.

Для корпоративных проектов, где одновременно работают большие команды, управляющие обширными потоками данных, DDD может стать важным стратегическим решением. Один из основных плюсов DDD – это упрощение масштабирования системы: когда бизнес растет, модель может адаптироваться к новым требованиям без радикальной перестройки. В корпоративных условиях, где риски при внедрении нового ПО и изменения существующего кода особенно высоки, DDD позволяет минимизировать возможные ошибки и снизить затраты на поддержку.

DDD помогает адаптировать систему под уникальные бизнес-процессы и требования конкретной компании. В больших проектах часто наблюдается высокая сложность взаимодействий между различными отделами и сервисами, и доменная модель помогает организовать их более гибко и точно. Также, благодаря антикоррупционным слоям, DDD позволяет легко интегрировать систему с устаревшими сервисами, что часто необходимо в корпоративных ИТ-инфраструктурах.

Несмотря на значительные преимущества, DDD не лишён сложностей. Одной из главных трудностей является высокая начальная

стоимость внедрения. Потребуется время и ресурсы на обучение сотрудников [Вахлова 2010], так как для успешного применения DDD разработчики и аналитики должны обладать глубокими знаниями предметной области. При этом подход может показаться чрезмерно сложным для небольших или простых проектов, где его полное внедрение будет неоправданным.

Кроме того, DDD требует активного участия бизнеса на всех стадиях проекта. Компании, где коммуникация между бизнесом и разработкой затруднена, могут столкнуться с тем, что сложность доменной модели станет препятствием для эффективного внедрения. Интеграция с устаревшими системами также может быть вызовом, так как данные, не всегда соответствующие требованиям DDD, усложняют разработку антикоррупционных слоёв [Бурдуков 2024].

DDD и Agile – подходы, которые могут дополнять друг друга, позволяя создавать гибкие и устойчивые системы. Agile ориентирован на итеративное развитие, постоянное улучшение и адаптацию системы к изменениям, и DDD хорошо вписывается в эти принципы, так как способствует адаптации системы к изменяющимся требованиям бизнеса. Важно, что обе методологии делают акцент на взаимодействие между командой разработки и бизнесом, что способствует взаимопониманию и снижению рисков при внесении изменений.

Однако DDD требует тщательного анализа предметной области для создания модели, чтобы в итоговой реализации информационная система не требовала увеличения технического обеспечения (масштабирования серверов, создания отдельной инфраструктуры для запуска проекта). Это может вступать в противоречие с Agile, когда итерации начинаются с минимального анализа и запускаются на раннем этапе. Совмещение этих подходов требует баланса: например, начинать проект с базовой модели, которую можно углублять и расширять в ходе итераций, одновременно с регулярными проверками согласованности доменной модели с актуальными требованиями бизнеса.

Несмотря на эффективность DDD, многие команды допускают ошибки при его внедрении. Над подобными решениями должны работать квалифицированные профессионалы. Одна из самых распространенных ошибок – излишнее моделирование, когда проект тратит ресурсы на проработку ненужных деталей доменной модели. Этот подход, известный как «Большой дизайн впереди» (Big Design Up Front), противоречит принципу адаптивного проектирования и делает систему слишком громоздкой и трудной для изменений. Очень важно не заниматься ремесленничеством, а применять DDD, когда есть или развиваются объективные обстоятельства к этому: перегруженная структура системы, или необходимость выдерживать высокие нагрузки.

Еще одной ошибкой является неправильное понимание общих принципов DDD, когда разработчики путают доменные модели с техническими компонентами системы. Важно понимать, что доменные модели должны полностью отражать логику предметной области, а не технические аспекты. Это требует плотного взаимодействия с бизнесом и правильного распределения задач между командой разработки и аналитиками.

В поисках примера внедрения DDD можно рассматривать финансовый сектор. Банки сталкиваются с необходимостью быстрой обработки транзакций и соблюдения регуляторных норм. Внедрение DDD позволило создать модели, которые точно отражают финансовые операции, автоматизировать процессы и упростить интеграцию с регуляторами. В качестве результата организации получили уменьшение времени обработки транзакций и повышение соответствия требованиям к спецификации и требованиям заказчика. Также в качестве примера внедрения DDD можно привести крупные сети магазинов, которые сталкиваются с изменениями в потребительских предпочтениях и необходимостью адаптации ассортимента. Внедрение DDD для создания гибких систем управления запасами и анализа данных о покупках позволило получить более точное прогнозирование спроса, улучшение управления ассортиментом и увеличение продаж.

С учетом современных технологий (например, микросервисов, облачных платформ и искусственного интеллекта) DDD продолжает развиваться, адаптируясь к новым требованиям. Доменные события и событийное взаимодействие (Event-Driven Design) становятся все более популярными, позволяя применять DDD в распределенных системах. Технологии продолжают влиять на подход DDD, и будущее показывает, что его роль будет расти в условиях гибридных архитектур и усложняющихся бизнес-процессов.

DDD становится все более актуальным в условиях быстро меняющегося рынка, где компании должны быстро адаптироваться к новым условиям, он подходит не всем проектам, но для систем, где важна точность в передаче бизнес-логики и долговременная поддержка, он может стать идеальным решением.

Библиографический список

Бурдуков В.П. Domain-Driven Design: преимущества и недостатки методологии, управляемой предметной областью // Научный аспект №7-2024. Информационные технологии. URL: https://na-journal.ru/7-2024-informacionnye-tekhnologii/14071-domain-driven-design-preimushchestva-i-nedostatki-metodologii-upravlyaemoi-predmetnoi-oblastyu (дата обращения: 15.11.2024).

Вахлова О. Программирование, управляемое предметной областью (DDD). СПб: БХВ-Петербург, 2010.

Вернон, Вон. Реализация методов проектно-ориентированного программирования. Диалектика, 2023. 688 с.

Дандан, Р. Методические материалы по формированию карты контекстов. // International journal of humanities and natural sciences, vol. 5-2, 2022.С. 21-25.

Эванс, Эрик. Domain-Driven Design: Проектирование, управляемое предметной областью. Структуризация сложных программных систем: Пер. с англ. М.: ООО «И.Д. Вильямс», 2017. 417 с.

GRNTI 20.53.01

DOMAIN-DRIVEN DESIGN: POSITIVE AND NEGATIVE ASPECTS OF PRACTICAL IMPLEMENTATION

A.A. Faitelson
2nd year student in Software technology and administration of information systems
Kursk State University
e-mail: z0tedd@gmail.com

Supervisor:
A.V. Krivonos
PhD in Technical Sciences, Associate Professor of the Department of Software and Administration of Information Systems
Kursk State University
e-mail: krivonos_av@kursksu.ru

The article is devoted to the consideration of data-oriented programming, which is becoming more and more topical nowadays. Its main aim is to abolish the gap between business and software development, allowing the teams to create accurate and easily maintained systems deeply integrated into business processes.

Key words: *Domain-Driven Design, DDD, design of information systems, architecture.*

References

Burdukov V.P. Domain-Driven Design: preimushchestva i nedostatki metodologii, upravlyaemoj predmetnoj oblast'yu // Nauchnyj aspekt №7-2024. Informacionnye tekhnologii. URL: https://na-journal.ru/7-2024-informacionnye-tekhnologii/14071-domain-driven-design-preimushchestva-i-nedostatki-metodologii-upravlyaemoi-predmetnoi-oblastyu (data obrashcheniya: 15.11.2024).

Vahlova O. Programmirovanie, upravlyaemoe predmetnoj oblast'yu (DDD). SPb: BHV-Peterburg, 2010.

Vernon, Von. Realizaciya metodov proektno-orientirovannogo programmirovaniya. Dialektika, 2023. 688 s.

Dandan, R. Metodicheskie materialy po formirovaniyu karty kontekstov. // International journal of humanities and natural sciences, vol. 5-2, 2022.S. 21-25.

Evans, Erik. Domain-Driven Design: Proektirovanie, upravlyaemoe predmetnoj oblast'yu. Strukturizaciya slozhnyh programmnyh sistem: Per. s angl. M.: OOO «I.D. Vil'yams», 2017. 417 s.