Предисловие

Вычисления показывают, что наша идея неосуществима.
Осталось только одно: осуществить ее.

Пьер-Жорж Латекоэр,
французский пионер воздухоплавания

Именно этим мы и займемся. Предметно-ориентированное проектирование программного обеспечения — слишком важное дело, чтобы оставить разработчиков без четких указаний по его успешной реализации.

Взлет-посадка

Когда я был ребенком, мой отец научился летать на небольших самолетах. Часто мы летали всей семьей. Иногда мы летали в другой аэропорт на обед, а потом возвращались. Когда у отца было меньше времени, но он хотел полетать, мы просто кружили над аэродромом, отрабатывая взлет и посадку.

Мы предпринимали и более долгие путешествия. В таких случаях у нас всегда была карта маршрута, которую папа рисовал заранее. Поскольку мы были еще детьми, нам поручали высматривать ориентиры на земле, чтобы не сбиться с пути. Это было очень увлекательно, потому что распознавать мелкие объекты на земле было довольно трудно. На самом деле я уверен, что папа всегда знал, где мы находились. У него на приборной панели были все средства ориентирования, и он имел лицензию на "слепой" полет.

Вид сверху изменил мои представления об окружающем мире. Время от времени папа и я пролетали над нашим сельским домом. Рассматривая дом с высоты в несколько сотен футов, я видел окрестности совсем по-другому. Когда папа кружил над домом, мама и мои сестры выбегали во двор, чтобы помахать нам. Я знал, что это были они, несмотря на то, что я не мог видеть их лица. Мы не могли разговаривать. Если бы я выкрикнул что-нибудь в окно самолета, то они не услышали бы меня. Я видел изгородь, отделяющую наш участок от дороги. На земле я мог бы пройти по нему, как по бревну. Сверху изгородь была похожа на тщательно переплетенные прутья. Там был огромный луг, который я каждое лето косил, проходя на косилке полосу за полосой. С воздуха я видел только море зеленого цвета, а не травинки.

Я любил эти моменты полета. Они врезались в мою память так, будто это было еще вчера. Но хотя мне очень нравилось летать, мне всегда хотелось на землю. И какими бы захватывающими ни были посадки с немедленным взлетом после

касания земли, они происходили слишком быстро, чтобы я мог почувствовать твердую почву под ногами.

Посадка с помощью DDD

Знакомство с предметно-ориентированным проектированием можно сравнить с ощущением ребенка от полета. Вид сверху — потрясающий, но иногда вещи выглядят настолько незнакомыми, что нетрудно потерять ориентиры. Перебраться из точки А в точку Б кажется нереальным. Людям, имеющим опыт работы в рамках DDD, всегда кажется, что они знают, где находятся. Они уже давно нарисовали маршрут и полностью доверяют своим навигационным приборам. Множество других людей чувствуют себя неуверенно. Им нужна возможность "приземлиться и пришвартоваться". Кроме того, им необходима карта, чтобы из точки, где они находятся, переместиться в точку, в которой они должны быть.

В книге *Предметно-ориентированное проектирование (DDD): структуризация сложных программных систем* [Эванс] Эрик Эванс сделал то, что называют бессмертной работой. Я твердо убежден, что эта книга будет настольной у разработчиков в течение многих десятилетий. Как и в других работах о шаблонах, автор поднял точку зрения на большую высоту, чтобы расширить горизонты. Однако, если мы хотим понять основы реализации DDD, возникают сложности. Кроме того, обычно мы хотим видеть больше подробных примеров. Вот если бы только мы могли приземлиться и побыть на земле немного дольше или даже просто поехать домой или в другое знакомое место!

Помимо прочего, я хотел мягко посадить читателей на землю, овладеть их самолетом и помочь вернуться домой по известному наземному маршруту. Это поможет вам понять методы реализации DDD на основе примеров, использующих знакомые инструменты и технологии. И так как ни один из нас не может оставаться дома все время, я также помогу вам путешествовать в места, где вы никогда не были, исследуя по пути новые ландшафты. Иногда путь будет крут, но, придерживаясь правильной тактики, можно безопасно взойти на гору. В этом восхождении вы узнаете об альтернативной архитектуре и шаблонах, предназначенных для интеграции многих моделей предметной области. Вы можете оказаться на неизвестной территории. Вы найдете подробное описание стратегического моделирования с многократной интеграцией и даже научитесь разрабатывать автономные службы.

Моя цель — предоставить карту, чтобы помочь вам осуществлять как короткие прогулки, так и длительные, сложные походы, получая удовольствие от окружающих вас деталей, не теряя ориентиров и не получая травм.

Изображение ландшафта и построение карты полета

Вполне очевидно, что при разработке программного обеспечения мы всегда связываем одни сущности с другими. Мы связываем объекты с базой данных или пользовательским интерфейсом, а также выполняем обратные действия. Мы связываем объекты с различными приложениями, включая те, которые могут быть использованы в других системах и приложениях. Исходя из этого, вполне естественно стремиться создать связь высокоуровневых шаблонов Эванса с реализацией.

Даже если вы уже несколько раз сталкивались с предметно-ориентированным проектированием, есть возможность извлечь из этого подхода дополнительную пользу. Иногда такой подход называют *облегченным DDD*. Мы сосредоточимся на ОБЪЕКТАХ (OBJECTS) и СЛУЖБАХ (SERVICES), возможно, предпримем смелую попытку разработать АГРЕГАТЫ (AGGREGATES) и попробуем управлять постоянным хранением АГРЕГАТОВ, используя ХРАНИЛИЩА (REPOSITORIES). Эти шаблоны всем знакомы, поэтому мы будем использовать именно их. Кроме того, попутно мы можем применить ОБЪЕКТЫ-ЗНАЧЕНИЯ (VALUE OBJECTS). Все эти шаблоны относятся к категории тактических шаблонов, которые носят скорее технический характер. Они помогают разбираться в серьезных проблемах, связанных с программным обеспечением, применяя навыки хирурга и скальпель. Однако нам надо еще очень многое узнать, чтобы овладеть этими и другими тактическими шаблонами. Я связываю их с реализацией.

Вы когда-нибудь выходили за пределы тактического моделирования? Вы когда-нибудь оказывались на "другой стороне" DDD, в области *стратегических проектных шаблонов*? Если вы никогда не использовали ОГРАНИЧЕННЫЙ КОНТЕКСТ (BOUNDED CONTEXT) и КАРТУ КОНТЕКСТА (CONTEXT MAP), то, вероятно, также ничего не знаете о шаблоне ЕДИНЫЙ ЯЗЫК (UBIQUITOUS LANGUAGE).

Если и существует единственное "изобретение", которое Эванс предоставил сообществу разработчиков программного обеспечения, это шаблон ЕДИНЫЙ ЯЗЫК. Как минимум он извлек этот шаблон из пыльных архивов премудростей проектирования. Это командный шаблон, который используется для фиксации понятий и терминов определенной предметной области в самой модели программного обеспечения. Модель программного обеспечения включает существительные, прилагательные, глаголы и более содержательные выражения, на которых формально говорит группа разработчиков, которая включает одного или нескольких экспертов в проблемной области. Однако было бы ошибкой считать, что единый язык ограничен только словами. Точно так же, как любой естественный язык отражает мышление тех, кто говорит на нем, единый язык отражает ментальную модель экспертов в предметной области, в которой вы работаете. Таким образом,

программное обеспечение и тесты, проверяющие соответствие модели принципам предметной области, фиксируют и твердо придерживаются правил языка, на котором думает и говорит команда проектировщиков. ЕДИНЫЙ ЯЗЫК так же ценен, как и другие стратегические и тактические шаблоны моделирования, а в некоторых случаях имеет дополнительные преимущества.

Проще говоря, применение облегченного подхода DDD приводит к созданию низкоуровневых моделей предметной области. Именно по этой причине шаблоны ЕДИНЫЙ ЯЗЫК, ОГРАНИЧЕННЫЙ КОНТЕКСТ и КАРТА КОНТЕКСТОВ являются такими многообещающими. Вы получаете больше, чем малопонятный жаргон команды. Язык команды в явном ограниченном контексте, выраженном как модель предметной области, добавляет истинную бизнес-ценность и дает нам уверенность, что мы реализуем корректное программное обеспечение. Даже с технической точки зрения это помогает нам создавать ясные модели с более мощными характеристиками, которые меньше уязвимы для ошибок. Таким образом, я связываю стратегические проектные шаблоны с понятными примерами реализации.

Эта книга отображает ландшафт DDD способом, который позволяет осознать преимущества как стратегического, так и тактического проектирования. Она связывает в одно целое бизнес-ценности и технические средства, глубоко погружая читателей в детали.

Было бы разочарованием, если бы все, что мы когда-либо делали с помощью подхода DDD, "осталось на земле". Застревая в деталях, мы забыли бы, что вид с высоты птичьего полета тоже многому учит. Не ограничивайтесь походами по пересеченной местности. Наберитесь смелости сесть в кресло пилота и посмотрите на проблему с высоты, о которой мы говорили. Учебные полеты с помощью стратегических шаблонов ОГРАНИЧЕННЫЙ КОНТЕКСТ и КАРТА КОНТЕКСТОВ дадут вам более широкое представление об их полноценной реализации. Когда вы совершите полет на самолете DDD, я буду считать, что достиг своей цели.

Обзор глав

Перейдем к обзору содержания глав и покажем, какую пользу из них можно извлечь.

Глава 1. Знакомство с DDD

В этой главе описаны преимущества подхода DDD и показано, как использовать большинство из них. Вы узнаете, какую пользу подход DDD может принести вашему проекту и вашей команде, когда вы сталкиваетесь со сложной системой. Вы научитесь оценивать проекты, чтобы выяснить, стоит ли применять к ним

подход DDD. Мы рассмотрим общепринятые альтернативы DDD и покажем, почему они часто приводят к проблемам. В главе закладываются основы DDD, поскольку в ней объясняется, как выполнить первые этапы проекта, и даже даются советы, как убедить руководство, экспертов в предметной области и членов команды разработчиков применить подход DDD. Все это позволит вам справиться с трудностями использования DDD.

Мы рассмотрим сценарий, описывающий исследование проекта, в котором участвуют гипотетическая компания и ее команда проектировщиков, сталкивающаяся с реальными проблемами DDD. Компания, заключившая контракт на создание инновационной продукции на основе платформы SaaS в многоарендной архитектуре, совершает многочисленные ошибки, характерные для первичного знакомства с подходом DDD, но делает важные открытия, помогающие решить проблемы и поддержать проект "на плаву". С проектом связано большинство разработчиков, поскольку для управления проектом используется приложение на основе методологии Scrum. Этот сценарий закладывает основы для следующих глав. Каждый стратегический и тактический шаблон описывается с точки зрения членов команды. В главе показано, как они делают свои ошибки и достигают успеха в освоении DDD.

Глава 2. ПРЕДМЕТНАЯ ОБЛАСТЬ, ПРЕДМЕТНАЯ ПОДОБЛАСТЬ И ОГРАНИЧЕННЫЕ КОНТЕКСТЫ

Что такое ПРЕДМЕТНАЯ ОБЛАСТЬ (DOMAIN), ПРЕДМЕТНАЯ ПОДОБЛАСТЬ (SUBDOMAIN) и СМЫСЛОВОЕ ЯДРО (CORE DOMAIN)? Что такое ОГРАНИЧЕННЫЙ КОНТЕКСТ? Как и почему его следует применять? Мы отвечаем на эти вопросы, анализируя ошибки, которые делает команда проектировщиков в нашем сценарии. На первом этапе своего проекта DDD они неправильно понимают ПОДОБЛАСТЬ, в которой они работают, ее ОГРАНИЧЕННЫЙ КОНТЕКСТ и ЕДИНЫЙ ЯЗЫК. Они практически ничего не знают о стратегическом проектировании и используют лишь тактические шаблоны для решения технических проблем. Это приводит к проблемам, связанным с первоначальной моделью предметной области. К счастью, они вовремя понимают, что произошло.

Главная мысль, которая проводится к этой главе, заключается в том, что применение ОГРАНИЧЕННОГО КОНТЕКСТА позволяет правильно разграничить и разделить модели. В ней не только анализируются распространенные ошибки при работе с шаблонами, но и даются советы по их эффективной реализации. В главе показано, как члены команды исправляют ошибки и в результате создают два разных ОГРАНИЧЕННЫХ КОНТЕКСТА. Это приводит к концепциям моделирования, правильно отделенным в третьем ОГРАНИЧЕННОМ КОНТЕКСТЕ, новому СМЫСЛОВОМУ ЯДРУ и основному примеру, используемому в книге.

Эта глава должна вызвать сильный интерес у читателей, испытывающих большие трудности, связанные с исключительно формальным применением подхода DDD. Если вы не знакомы со стратегическим проектированием, то эта глава покажет вам правильное направление.

Глава 3. КАРТЫ КОНТЕКСТОВ

Шаблон КАРТА КОНТЕКСТОВ — мощный инструмент, помогающий команде понять свою предметную область, очертить границы между разными моделями и выделить их реальную или потенциальную общность. Этот метод не сводится к рисованию диаграмм системной архитектуры. Его цель — понимание отношений между разными ОГРАНИЧЕННЫМИ КОНТЕКСТАМИ на предприятии и между шаблонами, используемыми для четкого отображения объектов одной модели в объекты другой модели. Этот инструмент играет важную роль для успешной работы с ограниченными контекстами на сложном предприятии. В главе описана работа команды проектировщиков, применяющих ОТОБРАЖЕНИЕ КОНТЕКСТА для понимания проблем, созданных их первым ОГРАНИЧЕННЫМ КОНТЕКСТОМ (глава 2). Затем в ней показано, как два итоговых ограниченных контекста влияют на проектирование и реализацию нового СМЫСЛОВОГО ЯДРА.

Глава 4. АРХИТЕКТУРА

Модель МНОГОУРОВНЕВАЯ АРХИТЕКТУРА (LAYERED ARCHITECTURE) знают все. Но являются ли уровни единственным способом описания приложения DDD или можно использовать другие архитектуры? В этой главе мы покажем, как использовать подход DDD в таких архитектурах, как ГЕКСАГОНАЛЬНАЯ (ПОРТЫ И АДАПТЕРЫ), СЕРВИСНО-ОРИЕНТИРОВАННАЯ, REST, CQRS, СОБЫТИЙНАЯ (КАНАЛЫ и ФИЛЬТРЫ, ДОЛГОВРЕМЕННЫЕ ПРОЦЕССЫ, или САГИ, ПОРОЖДЕНИЕ СОБЫТИЙ), а также DATA FABRIC/GRID-BASED. Некоторые из этих архитектур будут положены в основу проекта.

Глава 5. СУЩНОСТИ

Первым из тактических шаблонов DDD рассматривается шаблон СУЩНОСТЬ (ENTITY). Команда проекта слишком глубоко изучила этот шаблон, просмотрев возможность применить важный шаблон ОБЪЕКТ-ЗНАЧЕНИЕ (VALUE OBJECT). В результате возникла дискуссия о том, как избежать широко распространенного избыточного использования шаблона СУЩНОСТЬ, обусловленного чрезмерным влиянием баз данных и принципов постоянного хранения данных.

Научившись правильно применять этот шаблон, вы увидите множество примеров правильного проектирования сущностей. Как выразить ЕДИНЫЙ ЯЗЫК посредством СУЩНОСТЕЙ? Как тестировать, реализовывать и хранить СУЩНОСТИ? Мы ответим на каждый из этих вопросов.

Глава 6. ОБЪЕКТЫ-ЗНАЧЕНИЯ

На ранних этапах проектирования команда пропустила важные возможности моделирования с помощью шаблона ОБЪЕКТ-ЗНАЧЕНИЕ. Она слишком сосредоточилась на индивидуальных атрибутах СУЩНОСТЕЙ, не обратив внимания на то, что многочисленные связанные друг с другом атрибуты образуют некое неизменяемое целое. Проектирование на основе шаблона ОБЪЕКТ-ЗНАЧЕНИЕ рассматривается с разных точек зрения. В главе показано, как идентифицировать специальные характеристики модели, чтобы определить, какой шаблон использовать — ОБЪЕКТ-ЗНАЧЕНИЕ или СУЩНОСТЬ. Помимо этого, в главе рассматривается такая важная тема, как роль ОБЪЕКТОВ-ЗНАЧЕНИЙ в интеграции и моделировании СТАНДАРТНЫХ ТИПОВ (STANDARD TYPES). После этого показано, как проектировать предметно-ориентированные тесты, как реализовывать типы ЗНАЧЕНИЙ (VALUES) и как избежать вредного влияния механизмов постоянного хранения данных на хранение ЗНАЧЕНИЙ в качестве компонентов АГРЕГАТОВ.

Глава 7. СЛУЖБЫ

В главе показано, как распознать ситуации, в которых концепцию предметной области следует моделировать как мелкомодульную СЛУЖБУ (SERVICE), не имеющую состояния. Вы узнаете, когда следует проектировать СЛУЖБУ, а не СУЩНОСТЬ или ОБЪЕКТ-ЗНАЧЕНИЕ, и как реализовать СЛУЖБЫ ПРЕДМЕТНОЙ ОБЛАСТИ (DOMAIN SERVICES) для воплощения бизнес-логики и достижения целей технической интеграции. Описываются ситуации, в которых применяются службы, и демонстрируются методы их разработки.

Глава 8. СОБЫТИЯ ПРЕДМЕТНОЙ ОБЛАСТИ

В книге Эрика Эванса нет формального описания шаблона СОБЫТИЯ ПРЕДМЕТНОЙ ОБЛАСТИ (DOMAIN EVENTS). Вы узнаете, почему СОБЫТИЯ ПРЕДМЕТНОЙ ОБЛАСТИ, публикуемые моделью, являются такими мощными, а также как их можно использовать для достижения разных целей, в том числе для поддержки интеграции и автономных бизнес-служб. Несмотря на то что приложения посылают и обрабатывают разнообразные технические события, события предметной области обладают особенными характеристиками. В главе даются

рекомендации по проектированию и реализации, а также описываются существующие возможности и компромиссы. Затем показано, как создать механизм ИЗДАТЕЛЬ–ПОДПИСЧИК (PUBLISH-SUBSCRIBE), как публикуются события предметной области для подписчиков внутри предприятия, как создать ХРАНИЛИЩЕ СОБЫТИЙ (EVENT STORE) и управлять им, и как правильно решать типичные проблемы. Каждая из этих тем обсуждается в свете усилий команды проекта, направленных на стремление наилучшим образом использовать имеющиеся возможности.

Глава 9. МОДУЛИ

Как организовать объекты, существующие в рамках модели, в контейнеры правильного размера, имеющие ограниченное сцепление с объектами, находящимися в других контейнерах? Как назвать эти контейнеры, чтобы они отражали ЕДИНЫЙ ЯЗЫК? Как использовать более современные средства модульного проектирования, такие как OSGi и Jigsaw, поддерживаемые языками и каркасами? Здесь вы узнаете, как использовать МОДУЛИ (MODULES) в разных проектах.

Глава 10. АГРЕГАТЫ

АГРЕГАТЫ, вероятно, являются самым плохо понимаемым среди всех тактических шаблонов DDD. Впрочем, если принять определенные эмпирические правила, можно упростить шаблон АГРЕГАТЫ и облегчить его реализацию. Мы покажем, как преодолеть сложности использования шаблона АГРЕГАТ, создающего границы согласованности вокруг кластеров малых объектов. Обращая слишком большое внимание на относительно неважные аспекты АГРЕГАТОВ, команда проектировщиков в нашем сценарии столкнулась с разными препятствиями. Вместе с командой мы пройдем по нескольким путям, которые окажутся ошибочными, и покажем, как исправить ситуацию. В результате читатели должны более глубоко понять СМЫСЛОВОЕ ЯДРО. Мы увидим, как команда исправит свои ошибки, правильно применив концепцию транзакционной и конечной согласованности, и как она создаст более гибкую и эффективную модель в среде распределенной обработки.

Глава 11. ФАБРИКИ

В книге [Gamma et al.] много говорится о ФАБРИКАХ (FACTORIES), так зачем же писать о них еще раз? Это простая глава, в которой никто не пытается заново изобрести велосипед. Мы попытаемся понять, в каких ситуациях должны существовать ФАБРИКИ. Разумеется, есть хорошие рекомендации, указывающие, когда

следует проектировать ФАБРИКИ в рамках подхода DDD. Мы покажем, как наша команда создает ФАБРИКИ в рамках СМЫСЛОВОГО ЯДРА, чтобы упростить клиентский интерфейс и защитить пользователя модели от катастрофических ошибок в многоарендной среде.

Глава 12. ХРАНИЛИЩА

Является ли ХРАНИЛИЩЕ простым ОБЪЕКТОМ ДОСТУПА К ДАННЫМ (DATA ACCESS OBJECT — DAO)? А если нет, то в чем разница? Почему проектирование ХРАНИЛИЩ следует осуществлять как моделирование коллекций, а не баз данных? Мы покажем, как ХРАНИЛИЩЕ используется в сочетании с технологией ORM (Object Relational Mapping), поддерживающей связный решеточный распределенный кеш (coherence grid-based distributed cache) и использующей хранилище NoSQL "ключ–значение". Каждый из этих механизмов постоянного хранения данных стал доступен команде проекта благодаря мощи и универсальности шаблона ХРАНИЛИЩЕ.

Глава 13. Интеграция ОГРАНИЧЕННЫХ КОНТЕКСТОВ

После освоения высокоуровневых методов отображения контекстов и тактических шаблонов возникает вопрос "Как осуществить интеграцию моделей?" Какие возможности для интеграции предоставляет подход DDD? Глава раскрывает несколько способов интеграции с помощью шаблона КАРТА КОНТЕКСТОВ (CONTEXT MAP). Она содержит рекомендации по интеграции СМЫСЛОВОГО ЯДРА с другими вспомогательными ОГРАНИЧЕННЫМИ КОНТЕКСТАМИ, введенными в предыдущих главах.

Глава 14. Приложение

Мы разработали модель с помощью ЕДИНОГО ЯЗЫКА СМЫСЛОВОГО ЯДРА, а также примерные тесты для проверки ее работоспособности и правильности. А что делать другим членам команды проекта, разрабатывающим приложения, сопровождающие нашу модель? Должны ли они использовать ОБЪЕКТЫ ПЕРЕДАЧИ ДАННЫХ (DATA TRANSFER OBJECTS — DTO) для обмена информацией между моделью и пользовательским интерфейсом? Или они должны использовать другие возможности для передачи состояния модели компонентам представления? Как работают библиотека ПРИКЛАДНЫХ СЛУЖБ и инфраструктура? Ответы на эти вопросы, проиллюстрированные нашим учебным проектом, можно найти в этой главе.

Приложение. АГРЕГАТЫ И ИСТОЧНИКИ СОБЫТИЙ: A+ES

ИСТОЧНИКИ СОБЫТИЙ (EVENT SOURCING) — это важная техническая концепция хранения агрегатов, лежащая в основе разработки событийно-ориентированной архитектуры (Event-Driven Architecture). ИСТОЧНИКИ СОБЫТИЙ можно использовать для представления состояния агрегата в целом, которое является следствием последовательности событий, произошедших с момента его создания. События используются для восстановления состояния агрегата с помощью их воспроизведения в том же порядке, в котором они происходили. Предпосылкой для этого является тот факт, что такой подход упрощает обеспечение постоянного хранения данных и позволяет воплотить концепции со сложными поведенческими характеристиками. Кроме того, события сами по себе оказывают большое влияние пользовательские и внешние системы.

Язык Java и инструменты разработки

В большинстве примеров в книге использован язык Java. Я мог бы привести примеры на языке C#, но пришел к выводу, что следует использовать Java.

Во-первых, как это ни досадно, я считаю, что члены сообщества Java пренебрегают хорошими методами проектирования и разработки. В настоящее время сложно найти ясную и четкую модель предметной области в большинстве проектов на языке Java. Мне кажется, что методология Scrum и другие технологии гибкого проектирования вытеснили тщательное моделирование, а разработчикам внушают, что список заданий (product backlog) — это всего лишь набор проектов. Большинство проектировщиков, использующих методы гибкой разработки, мало задумываются о том, как их задания повлияют на базовую бизнес-модель. По-моему очевидно, что методология Scrum, например, никогда не предназначалась для замены проектирования. Независимо от того, сколько менеджеров по проектам и продукции хотели бы видеть вас бодро марширующими на жестоком пути непрерывного развертывания (continuous delivery), технология Scrum — это не просто средство доставить удовольствие энтузиастам диаграмм Ганта (Gantt chart), хотя во многих случаях она используется только для этого.

Я считаю, что это большая проблема, и хотел бы вернуть сообщество Java к моделированию предметной области, заставив его подумать о том, насколько большую пользу могут им принести солидные, а не гибкие и быстрые методы проектирования.

Использованию подхода DDD на платформе .NET посвящено достаточное количество хороших книг, например книга Jimmy Nilsson *Applying Domain-Driven Design and Patterns: With Examples in C# and .NET* [Nilsson]. Благодаря книге Джимми и деятельности других специалистов, продвигающих принципы Alt.NET, в

сообществе .NET появилось много хороших методов проектирования и разработки. Разработчики на языке Java должны принять это к сведению.

Во-вторых, мне хорошо известно, что сообщество C#.NET не имеет проблем с пониманием кода на языке Java. Поскольку большинство членов сообщества DDD использует язык C#.NET и большинство рецензентов предыдущего издания были программистами на C#, я не получал никаких жалоб, касавшихся понимания кода на языке Java. Итак, я убежден, что использование языка Java в книге никоим образом не оттолкнет программистов на языке C#.

Следует добавить, что пока я писал книгу, произошел значительный сдвиг от реляционных баз данных в сторону документных хранилищ и хранилищ "ключ–значение". Это настолько серьезное явление, что Мартин Фаулер (Martin Fowler) быстро подобрал для них общее название "агрегатно-ориентированное хранилище". Это подходящее название, которое хорошо описывает преимущества использования хранилищ NoSQL в проектах DDD.

В ходе моей консультационной деятельности я убедился, что многие по-прежнему остаются приверженцами реляционных баз данных и объектно-реляционного отображения. По этой причине я полагаю, что с практической точки зрения энтузиастам NoSQL было бы полезно последовать моим советам, касающимся методов объектно-реляционного отображения для моделей предметной области. Однако я знаю, что это может вызвать ко мне презрение со стороны тех, кто думает, что объектно-реляционное рассогласование (impedance mismatch) не заслуживает внимания. Отлично, я принимаю огонь на себя, потому что существует огромное множество людей, которые должны как-то ежедневно справляться с этим рассогласованием, вызывая возможные насмешки со стороны меньшинства.

Кроме того, в главе 12, "Хранилища", я даю рекомендации по использованию документных хранилищ, хранилищ "ключ–значение" и хранилищ тип Data Fabric/Grid-Based. В нескольких местах я обсуждаю, как использование хранилища NoSQL могло бы повлиять на альтернативный проект агрегатов и их компонентов. Вероятно, тенденция к интенсивному использованию хранилищ NoSQL в этом секторе продолжится, и разработчики объектно-реляционных проектов должны учесть это обстоятельство. Как видите, я понимаю аргументы обеих сторон и с обеими согласен. Все это часть продолжающихся споров, вызванных технологическими тенденциями, и эти споры необходимы, чтобы позитивные изменения продолжались.