ПРЕДМЕТНО-ОРИЕНТИРОВАННОЕ ПРОЕКТИРОВАНИЕ

САМОЕ ОСНОВНОЕ

ВОН ВЕРНОН

Предметно-ориентированное проектирование

САМОЕ ОСНОВНОЕ

Domain-Driven Design Distilled

Vaughn Vernon

Addison-Wesley

Boston • Columbus • Indianapolis • New York • San Francisco • Amsterdam • Cape Town
Dubai • London • Madrid • Milan • Munich • Paris • Montreal • Toronto • Delhi • Mexico City
São Paulo • Sydney • Hong Kong • Seoul • Singapore • Taipei • Tokyo

Предметно-ориентированное проектирование

САМОЕ ОСНОВНОЕ

Вон Вернон

Москва • Санкт-Петербург • Киев
2017

ББК 32.973.26-018.2.75
В35
УДК 681.3.07

Компьютерное издательство "Диалектика"
Зав. редакцией *С.Н. Тригуб*
Перевод с английского и редакция докт. физ.-мат. наук *Д.А. Клюшина*
По общим вопросам обращайтесь в издательство "Диалектика" по адресу:
info@dialektika.com, http://www.dialektika.com

Вернон, Вон.
В35 Предметно-ориентированное проектирование: самое основное. : Пер. с англ. — СпБ. : ООО "Альфа-книга", 2017. — 160 с. : ил. — Парал. тит. англ.
ISBN 978-5-9908463-8-8 (рус.)

ББК 32.973.26-018.2.75

Все названия программных продуктов являются зарегистрированными торговыми марками соответствующих фирм.

Никакая часть настоящего издания ни в каких целях не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами, будь то электронные или механические, включая фотокопирование и запись на магнитный носитель, если на это нет письменного разрешения издательства Addison-Wesley Publishing Company, Inc.

Authorized translation from the English language edition published by Addison-Wesley Publishing Company, Inc. © 2016 by Pearson Education, Inc.

All rights reserved. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise.

Rights to this book were obtained by arrangement with Pearson Education, Inc.

Russian language edition published by Dialektika Computer Books Publishing according to the Agreement with R&I Enterprises International, Copyright © 2017

Научно-популярное издание

Вон Вернон

Предметно-ориентированное проектирование самое основное

Литературный редактор *И.А. Попова*
Верстка *М.А. Удалов*
Художественный редактор *В.Г. Павлютин*
Корректор *Л.А. Гордиенко*

Отпечатано в АО «Первая Образцовая типография»
Филиал «Чеховский Печатный Двор»
142300, Московская область, г. Чехов, ул. Полиграфистов, д.1
Сайт: www.chpd.ru, E-mail: sales@chpd.ru, тел. 8(499)270-73-59

ООО "Альфа-книга", 195027, Санкт-Петербург, Магнитогорская ул., д. 30

ISBN 978-5-9908463-8-8 (рус.)
ISBN 978-0-13-443442-1 (англ.)

© Компьютерное издательство "Диалектика", 2017
© Pearson Education, Inc., 2016

Оглавление

Предисловие 10

Благодарности 14

Об авторе 15

Глава 1

Краткий обзор DDD 17

Глава 2

Стратегическое проектирование с помощью ОГРАНИЧЕННЫХ КОНТЕКСТОВ и ЕДИНОГО ЯЗЫКА 27

Глава 3

Стратегическое проектирование с помощью ПОДОБЛАСТЕЙ 61

Глава 4

Стратегическое проектирование на основе СВЯЗЫВАНИЯ КОНТЕКСТОВ 67

Глава 5

Тактическое проектирование с помощью АГРЕГАТОВ 89

Глава 6

Тактическое проектирование с помощью СОБЫТИЙ ПРЕДМЕТНОЙ ОБЛАСТИ 113

Глава 7

Инструментальные средства для повышения эффективности проектирования 125

Библиография 151

Предметный указатель 153

Содержание

Предисловие 10
Для кого предназначена эта книга 11
Темы, рассмотренные в книге 12
Соглашения 13
Благодарности 14
Об авторе 15

Глава 1
Краткий обзор DDD 17
Вреден ли DDD? 18
Хороший, плохой и эффективный дизайн 19
Стратегическое проектирование 23
Тактическое проектирование 24
Процесс обучения и уточнения знаний 25
Поехали! 26

Глава 2
Стратегическое проектирование с помощью ОГРАНИЧЕННЫХ КОНТЕКСТОВ и ЕДИНОГО ЯЗЫКА 27
Эксперты проблемной области и бизнес-факторы 34
Типичный пример 37
Необходимость фундаментального стратегического проектирования 41
Ставьте проблемы и обобщайте 45
Разработка ЕДИНОГО ЯЗЫКА 51
Реализация сценариев 55
Далекие перспективы 57
Архитектура 57
Резюме 59

Глава 3
Стратегическое проектирование с помощью ПОДОБЛАСТЕЙ 61
Что такое ПОДОБЛАСТЬ 62
Типы ПОДОБЛАСТЕЙ 62
Проблема сложности системы 63
Резюме 66

Глава 4

Стратегическое проектирование на основе связывания контекстов 67

Способы связывания контекстов 69

ПАРТНЕРСТВО 70

ОБЩЕЕ ЯДРО 70

КЛИЕНТ-ПОСТАВЩИК 71

КОНФОРМИСТ 71

ПРЕДОХРАНИТЕЛЬНЫЙ УРОВЕНЬ 72

СЛУЖБА С ОТКРЫТЫМ ПРОТОКОЛОМ 73

ОБЩЕДОСТУПНЫЙ ЯЗЫК 73

ОТДЕЛЬНОЕ СУЩЕСТВОВАНИЕ 74

БОЛЬШОЙ КОМ ГРЯЗИ 74

Правильное использование СВЯЗЫВАНИЯ КОНТЕКСТОВ 76

Удаленный вызов процедур по протоколу SOAP 77

Протокол RESTful HTTP 78

РАССЫЛКА СООБЩЕНИЙ 80

Пример СВЯЗЫВАНИЯ КОНТЕКСТОВ 85

Резюме 88

Глава 5

Тактическое проектирование с помощью АГРЕГАТОВ 89

Зачем нужны АГРЕГАТЫ 90

Эмпирические правила проектирования АГРЕГАТОВ 95

Правило 1. Защищайте бизнес-инварианты в границах АГРЕГАТА 95

Правило 2. Проектируйте маленькие АГРЕГАТЫ 97

Правило 3. Ссылайтесь на другие АГРЕГАТЫ только по идентификаторам 98

Правило 4. Обновляйте другие АГРЕГАТЫ, руководствуясь принципом итоговой согласованности 99

Моделирование агрегатов 102

Тщательно выбирайте абстракции 107

Агрегаты правильного размера 109

Тестируемые модули 111

Резюме 112

Глава 6

Тактическое проектирование с помощью СОБЫТИЙ ПРЕДМЕТНОЙ ОБЛАСТИ 113

Проектирование, реализация и использование СОБЫТИЙ ПРЕДМЕТНОЙ ОБЛАСТИ 114

ИСТОЧНИКИ СОБЫТИЙ 121
Резюме 123

Глава 7

Инструментальные средства для повышения эффективности проектирования 125
СОБЫТИЙНЫЙ ШТУРМ 126
Другие инструменты 138
Применение принципов DDD для гибкого проектирования 138
Начнем с начала 140
Использование SWOT-анализа 140
Всплески и долги моделирования 142
Идентификация задач и оценивание 143
Моделирование с ограничением времени 145
Реализация 146
Взаимодействие с ЭКСПЕРТАМИ ПРЕДМЕТНОЙ ОБЛАСТИ 148
Резюме 149

Библиография 151

Предметный указатель 153

Посвящается Николь и Тристан!
Мы сделали это снова!

Предисловие

Почему моделирование — такое увлекательное и приятное занятие? Еще в детстве я любил конструировать модели. Тогда я собирал, как правило, модели автомобилей и самолетов. Я не уверен, что конструктор LEGO уже существовал в те дни. Однако LEGO был важной частью жизни моего сына, когда он был маленьким. Конструировать и собирать модели из маленьких кубиков было очень интересно. Сначала вы строите элементарные модели, а потом вам кажется, что ваши идеи можно развивать до бесконечности. Вероятно, каждый человек в детстве проходил через это увлечение.

Модели возникают в очень многих жизненных ситуациях. Если вы любите играть в настольные игры, то используете модели. Это могут быть модели недвижимого имущества и его владельцев, или островов и выживших людей, или территорий и строительства, и кто знает, что еще. Точно так же видеоигры — это модели. Они моделируют сказочный мир с причудливыми персонажами, играющими фантастические роли. Даже колода карт и карточные игры моделируют власть. Мы настолько часто используем модели, что, как правило, не осознаем этого. Модели — это часть нашей жизни.

Но почему? У каждого человека есть свои особенности восприятия. Существует множество видов восприятия, но среди них есть три главные — слух, зрение и осязание. Люди, воспринимающие реальность через слух, познают мир, слыша и слушая. Люди, у которых превалирует зрительное восприятие, обучаются, читая или видя образы. Люди, воспринимающие реальность через осязание, получают знания, прикасаясь к чему-то. Интересно, что у каждого человека один из этих видов восприятия господствует над остальными до такой степени, что он может иногда испытывать трудности с другими типами восприятия. Например, люди, воспринимающие реальность с помощью осязания, могут вспомнить, что они делали, но могут не вспомнить, что при этом говорили окружающие. Может показаться, что при разработке моделей люди, воспринимающие мир с помощью зрения и осязания, должны иметь огромное преимущество перед людьми, познающими мир с помощью слуха, потому что разработка модели кажется главным образом связанной со стимуляцией зрения и осязания. Однако это не всегда так, особенно, если группа разработчиков моделей в процессе работы использует звуковые средства связи. Иначе говоря, разработка моделей доступна для людей с любыми особенностями восприятия.

С нашей врожденной склонностью к восприятию реальности с помощью создания моделей совершенно естественно возникает желание моделировать программное обеспечение, которое играет все бо́льшую роль в нашей

жизни. Моделировать программное обеспечение — нормальное занятие для человека. И мы должны это делать. Мне кажется, что все люди могут быть прекрасными разработчиками моделей программного обеспечения.

Я очень хочу помочь читателям, насколько это возможно, проявить свои лучшие качества в моделировании программного обеспечения с помощью самых эффективных из доступных средств *предметно-ориентированного проектирования*, или *DDD* (domain-driven design). Этот набор инструментов, представляющих собой совокупность шаблонов, впервые был проанализирован Эриком Эвансом (Eric Evans) в книге *Domain-Driven Design: Tackling Complexity in the heart of Software*[1]. Я бы хотел, чтобы принципы DDD освоил каждый разработчик. Если это значит, что я хочу внедрить принципы DDD в массы, пусть так и будет. DDD заслуживает этого. DDD — это инструментарий, которые люди имеют право использовать для создания самых сложных моделей программного обеспечения. Я написал эту книгу, чтобы сделать изучение и использование DDD максимально простым и доступным для самой широкой аудитории.

Для людей, воспринимающих мир с помощью слуха, DDD открывает возможность обучения на основе общения в группе разработчиков модели, создающих ЕДИНЫЙ ЯЗЫК (UBIQUITOUS LANGUAGE). Люди, воспринимающие реальность с помощью зрения и осязания, оценят визуальный и тактильный характер процесса использования инструментальных средств DDD для стратегического и тактического проектирования. Это особенно ярко проявляется при создании КАРТ КОНТЕКСТОВ (CONTEXT MAPS) и моделировании бизнес-процессов с помощью СОБЫТИЙНОГО ШТУРМА (EVENT STORMING). Таким образом, я полагаю, что DDD может удовлетворить потребности каждого, кто хочет учиться и достичь успеха с помощью разработки моделей.

Для кого предназначена эта книга

Эта книга предназначена для тех, кто хочет быстро изучить самые важные аспекты и инструменты DDD. Целевая аудитория этой книги — архитекторы и разработчики программного обеспечения, желающие внедрить принципы DDD в свои проекты. Очень часто разработчики программного обеспечения быстро распознают преимущества DDD и оценивают по достоинству его мощь. Несмотря на это, я стремился раскрыть эту тему и для остальной аудитории — руководителей, экспертов предметной области, менеджеров, бизнес-аналитиков, архитекторов информационных систем и

[1] Русский перевод: Эванс Э. *Предметно-ориентированное проектирование (DDD): структуризация сложных программных систем.* — М.: Вильямс, 2011. — 444 с. — *Примеч. ред.*

тестировщиков. Пользу от чтения этой книги могут получить все, кто связан с информационными технологиями и научно-исследовательскими или конструкторскими проектами.

Если вы — консультант и работаете с клиентом, которому рекомендуете использовать DDD, то используйте эту книгу как средство, позволяющее быстро донести основные идеи до заинтересованных сторон. Если над вашим проектом работают разработчики младшего или среднего, а может быть, и старшего уровня, которые не знают DDD, но должны его очень быстро освоить, посоветуйте им эту книгу. Как минимум, прочитав эту книгу, все заинтересованные стороны и разработчики будут использовать одни и те же термины и знать основные инструменты DDD. Это даст им возможность осознанно использовать DDD в ходе совместной работы на проектом.

Независимо от вашего опыта и роли в проекте, прочитайте эту книгу, а затем примените DDD на практике. После этого снова перечитайте книгу и выясните, что оказалось полезным и как это можно улучшить.

Темы, рассмотренные в книге

В главе 1, "Краткий обзор DDD", я объясняю, чем DDD может быть полезным для вас и вашей организации, а также привожу подробный, но краткий обзор того, чему вы будете учиться и почему это важно для вас.

Глава 2, "Стратегическое проектирование с помощью ОГРАНИЧЕННЫХ КОНТЕКСТОВ и ЕДИНОГО ЯЗЫКА", посвящена вопросам стратегического предметно-ориентированного проектирования на основе его краеугольных камней: ОГРАНИЧЕННЫХ КОНТЕКСТОВ и ЕДИНОГО ЯЗЫКА. В главе 3, "Стратегическое проектирование с помощью ПОДОБЛАСТЕЙ", объясняется концепция ПОДОБЛАСТЕЙ и способы ее использования для упрощения интеграции унаследованных систем с новыми приложениями. Глава 4, "Стратегическое проектирование на основе СВЯЗЫВАНИЯ КОНТЕКСТОВ", описывает разнообразие способов организации стратегического сотрудничества групп и объединения их программного обеспечения. Эта концепция называется СВЯЗЫВАНИЕМ КОНТЕКСТОВ. Глава 5, "Тактическое проектирование с помощью АГРЕГАТОВ," переключает ваше внимание на тактическое моделирование и АГРЕГАТЫ. Важный и мощный тактический инструмент моделирования, который используется вместе с АГРЕГАТАМИ, — СОБЫТИЯ ПРЕДМЕТНОЙ ОБЛАСТИ, являются темой главы 6, "Тактическое проектирование с помощью СОБЫТИЙ ПРЕДМЕТНОЙ ОБЛАСТИ". Наконец, в главе 7, "Инструментальные средства для повышения эффективности проектирования", описываются инструменты, предназначенные для ускорения и управления проектами, которые могут помочь группам установить и поддерживать правильный темп работы. Эти

две темы редко обсуждаются в других источниках по DDD, но они определенно необходимы для тех, кто настроен внедрить DDD в практику.

Соглашения

В книге принято лишь несколько соглашений. Все шаблоны DDD, которые мы будем обсуждать, набраны ПРОПИСНЫМИ БУКВАМИ. Например, вы будете читать об ОГРАНИЧЕННЫХ КОНТЕКСТАХ и СОБЫТИЯХ ПРЕДМЕТНОЙ ОБЛАСТИ. Другое соглашение касается фрагментов кода — они выделены моноширинным шрифтом. Аббревиатуры источников, которые приведены в библиографии, указаны в главах в квадратных скобках.

Кроме того, в книге особое внимание уделяется многочисленным диаграммам и рисункам, которые обычно воспринимаются читателями лучше, чем текст. Обратите внимание на то, что рисунки не пронумерованы, потому что я не хотел смущать читателей их большим количеством. Рисунки и диаграммы всегда предшествуют их обсуждению в тексте, так что определенный визуальный образ постоянно будет сопровождать вас при чтении. Это означает, что, читая текст, вы можете вернуться к предыдущему рисунку или диаграмме для визуальной поддержки.

Благодарности

Это уже третья моя книга, опубликованная уважаемым издательством Addison-Wesley. Кроме того, я в третий раз работаю с моим редактором Крисом Гузиковски (Chris Guzikowski) и техническим редактором Крисом Заном (Chris Zan). Я счастлив сказать, что третий раз был таким же приятным, как первые два. Еще раз спасибо за выбор моей книги для публикации.

Ни одна книга не может быть успешно написана и издана без критических замечаний. На сей раз я обратился к практикам DDD. Это не всегда были преподаватели или авторы книг и статей, но эти люди играли важную роль в проектах, помогая другим использовать мощный инструментарий DDD. Я считал, что именно практики должны подтвердить необходимость и правильность изложенного материала. Как говорится, если вы хотите, чтобы я выступал в течение 60 минут, дайте мне 5 минут на подготовку, а если вы хотите, чтобы я выступал в течение 5 минут, дайте мне на подготовку несколько часов.

Далее в алфавитном порядке перечислены те, кто помогал мне больше всех: Джереми Чассен (Jérémie Chassaing), Брайен Данлэп (Brian Danlap), Юджи Кирики (Yuji Kiriki), Том Стоктон (Tom Stockton), Тормод Дж. Вархаугвик (Tormod J. Varhaugvik), Даниэль Вестхайде (Daniel Westheide) и Филип Уиндли (Philip Windley). Большое спасибо!

Об авторе

Вон Вернон — ветеран программирования и авторитетный эксперт в области упрощения проектирования и реализации программного обеспечения. Он автор бестселлера *Implementing Domain-Driven Design and Reactive Messaging Patterns with the Actor Model*, также опубликованного в издательстве Addison–Wesley. Он организовал семинар IDDD Workshop, который проводится по всему миру для сотен разработчиков программного обеспечения. Вон часто выступает на профессиональных конференциях. Он больше всего интересуется распределенными вычислениями, передачей сообщений и, в особенности, моделью акторов. Вон специализируется на консультациях по предметно-ориентированному проектированию, используя модель акторов вместе с языком Scala и каркас Akka. Вы можете следить за работами Вона, читая его блог по адресу `www.VaughnVernon.co` и сообщения в Твиттере по адресу `@VaughnVernon`

Глава 1

Краткий обзор DDD

Я уверен, что вы хотите улучшить свое мастерство и закрепить успех своих проектов. Вы стремитесь повысить конкурентоспособность своего бизнеса за счет нового программного обеспечения, которое разрабатываете. Вы хотите реализовать программное обеспечение, которое не только правильно моделирует операции вашего бизнеса, но и обеспечивает масштабирование на основе самой современной программной архитектуры. Для того чтобы достичь этих целей, вам необходимо быстро освоить основы предметно-ориентированного проектирования (DDD).

DDD — это инструментарий, помогающий проектировать и реализовывать программное обеспечение, приносящее большую пользу как с тактической, так и стратегической точек зрения. Ваш организация не может быть лучшей во всем, следовательно, необходимо тщательно выбрать области, в которых она должна быть лучше других. Инструменты стратегического предметно-ориентированного проектирования помогут вам и вашей группе принимать наиболее эффективные решения, связанные с разработкой и интеграцией программного обеспечения. Ваша организация извлечет максимальную выгоду из программных моделей, которые явно отражают ее специализацию. Средства тактического предметно-ориентированного проектирования могут помочь вам и вашей группе разрабатывать максимально полезное программное обеспечение, точно моделирующее уникальные бизнес-операции. Ваша организация должна извлечь выгоду из широкого спектра возможностей, связанных с развертыванием решений в разнообразных инфраструктурах как внутри компании, так и в облаке. С помощью DDD вы и ваша группа можете применять самые эффективные методы проектирования и реализации программного обеспечения, необходимого для преуспевания в современной конкурентной бизнес-среде.

В этой книге я излагаю основы DDD, рассматривая как стратегические, так и тактические инструменты моделирования. Я понимаю уникальные требования к разработке программного обеспечения и проблемы, с которыми вы сталкиваетесь в ходе работы, стремясь улучшить свое положение в быстро изменяющейся отрасли промышленности. Вы не можете тратить месяцы на чтение книг по DDD и хотите освоить его, чтобы как можно скорее применить на практике.

Я — автор бестселлера *Implementing Domain-Driven Design* [IDDD][1]. Кроме того, я организовал и провожу трехдневный семинар IDDD Workshop. Я написал эту книгу, чтобы изложить принципы DDD в максимально сжатом виде. Это мой вклад в дело внедрения DDD в практику каждой группы разработчиков программного обеспечения, где это уместно. Моя цель, конечно, — объяснить вам основы DDD.

Вреден ли DDD?

Возможно, вы слышали, что DDD — сложный подход к разработке программного обеспечения. Сложный? Он не сложнее задач, для решения которых предназначен. Действительно, DDD — это совокупность передовых методик, которые используются при разработке сложного программного обеспечения. С учетом мощи и масштаба этого подхода для его самостоятельного изучения без помощи эксперта вам потребуется так много усилий, что это может отпугнуть вас от его освоения. Вы, вероятно, также думаете,

[1] Русский перевод: Вернон В. *Реализация методов предметно-ориентированного проектирования.* — М.: Вильямс, 2016. — 688 с. — *Примеч. ред.*

что другие книги по DDD, содержащие сотни страниц, очень трудно понять и применить полученные навыки на практике. Мне пришлось затратить много слов, чтобы подробно объяснить DDD и дать его исчерпывающее описание, осветив более десятка специальных тем и шаблонов. В результате этих усилий в свет вышла книга *Implementing Domain-Driven Design* [IDDD]. Эта новая лаконичная книга должна ознакомить вас с самыми важными частями DDD так быстро и просто, насколько это возможно. Почему? Потому что многих читателей угнетают длинные тексты. Им нужны краткие справочники, чтобы немедленно приступить к освоению новых методов. Я выяснил, что специалисты, использующие DDD, перечитывают книги по нескольку раз. Вы можете подумать, что полностью изучить DDD невозможно, и поэтому станете использовать эту книгу как справочник, обращаясь за подробным описанием к другим источникам по мере углубления ваших знаний. Другие уже столкнулись с проблемами, пытаясь объяснить DDD своим коллегам и менеджерам. Эта книга поможет вам сделать это, не только объясняя DDD в сжатом виде, но и демонстрируя доступные инструментальные средства, позволяющие ускорить проектирование и обеспечить его управляемость.

Конечно, по этой книге нельзя пройти полный курс по DDD, потому что я преднамеренно изложил методики DDD в сжатом виде. Для более глубокого изучения этого подхода обратитесь к моей книге *Implementing Domain-Driven Design* [IDDD] или запишитесь на трехдневный семинар IDDD Workshop. Трехдневный интенсивный курс, который я провожу по всему миру для широкой аудитории, состоящей из сотен разработчиков, позволит вам быстро войти в курс дела. Кроме того, я также провожу обучение методам DDD в интерактивном режиме на сайте `http://ForComprehension.com`.

Хорошие новости заключаются в том, что DDD не должен повредить вам. Поскольку вы, вероятно, уже столкнулись со сложностью в ваших проектах, вы можете учиться использовать DDD, чтобы преодолеть сложность с помощью менее болезненных средств.

Хороший, плохой и эффективный дизайн

Люди часто говорят о хорошем и плохом дизайне. К какой категории проектов относится то, что делаете вы? Многие группы разработчиков программного обеспечения даже не думают о дизайне. Вместо этого они делают то, что я называю “перетасовкой задач на доске” (“taskboard shuffle”). Члены группы составляют список задач, связанных с разработкой, вроде журнала пожеланий продукта Scrum, или бэклога продукта (backlog), и перемещают стикер из столбца “В планах” в столбец “В работе”. Разработчики

глубокомысленно перетасовывают пункты списка в бэклоге продукта, а программисты тем временем все это героически кодируют. Результат редко совпадает с ожидаемым, а бизнес обычно платит самую высокую цену за отсутствие проектирования.

Это часто происходит из-за слишком напряженного календарного плана выпусков программного обеспечения, когда менеджеры используют методологию Scrum в основном для контроля за графиком работ и пренебрегают одним из самых важных принципов Scrum: *приобретением знаний*.

Когда я консультирую или преподаю в отдельных компаниях, я часто сталкиваюсь с одной и той же ситуацией: программный проект находится под угрозой, и все группы разработчиков нацелены на обеспечение работоспособности системы с помощью ежедневного исправления кода и данных. Ниже перечислены некоторые из коварных проблем, которые подстерегают разработчиков и которые DDD позволяет избежать. Я начинаю с бизнес-проблем высокого уровня и заканчиваю более техническими задачами.

- Разработка программного обеспечения считается статьей расходов, а не источником прибыли. Это объясняется тем, что бизнес часто рассматривает компьютеры и программное обеспечение как необходимый атрибут, а не источник стратегических преимуществ. (К сожалению, если такая бизнес-культура глубоко укоренилась в компании, то, вероятно, эту ситуацию исправить невозможно.)
- Разработчики слишком погружены в технические вопросы и пытаются устранять проблемы, используя технологию, а не трезвый расчет и продуманный дизайн. В результате они постоянно гоняются за новыми "блестящими штучками" — последними причудами технологии.
- Базе данных присваивается слишком высокий приоритет, и большинство обсуждений возможных решений сосредоточивается на базе данных и модели данных вместо бизнес-процессов и операций.
- Разработчики не придают надлежащего значения правильному наименованию объектов и операций в соответствии с бизнес-целью, для которой они предназначены. Это ведет к большой пропасти между ментальной моделью бизнес-экспертов и программным обеспечением, которое поставляют разработчики.
- Предыдущая проблема является следствием плохого сотрудничества с бизнес-экспертами. Часто бизнес-эксперты тратят слишком много времени на обособленную работу над спецификациями, которые либо не используются разработчиками вообще, либо учитываются частично.
- Слишком большое значение приписывается сметам проектов, на создание которых тратятся значительные усилия и объемы време-

ни, что приводит к задержке поставок программного обеспечения. Вместо продуманного проектирования разработчики перетасовывают задачи на доске. Они создают БОЛЬШОЙ КОМ ГРЯЗИ (обсуждаемый в следующих главах) вместо правильного разделения модели в соответствии с бизнес-факторами.

- Разработчики помещают бизнес-логику в компоненты пользовательского интерфейса и хранения данных. Кроме того, они часто выполняют операции, связанные с хранением данных в бизнес-логике.
- В системах возникают испорченные, медленные и вызывающие блокировку запросы к базе данных, не позволяющие пользователям выполнять бизнес-операции, чувствительные к задержкам.
- Разработчики используют неправильные абстракции, пытаясь учесть все существующие и предполагаемые потребности, чрезмерно обобщая решение, вместо того, чтобы обратиться к конкретным бизнес-потребностям.
- В системе используются сильно связанные службы, в которых операция выполняется одной службой, а затем эта служба непосредственно вызывает другую службу для выполнения балансирующей операции. Эта связь часто разрушает бизнес-процессы и порождает несогласованные данные, не говоря уже о том, что такие системы очень трудно обслуживать.

Все это кажется результатами отказа от проектирования с целью удешевления программного обеспечения. Слишком часто это является следствием неправильных бизнес-решений, принятых руководством компании и разработчиками программного обеспечения, которые не знают о существовании намного лучшей альтернативы. "Программное обеспечение пожирает мир" [WSJ], а значит, и вашу прибыль.

Важно понять что предполагаемая экономия за счет отказа от проектирования — это ложная идея, вводящая в заблуждение тех, кто настаивает на производстве программного обеспечения без продуманного дизайна. Все это происходит потому, что понимание важной роли проектирования все еще является достоянием немногих людей, и пока не оказывает влияния на остальных людей, включая бизнесменов. Я думаю, итог следует подвести с помощью следующей цитаты.

> *Вопросы о том, необходим ли дизайн или доступен, не имеют смысла: дизайн неизбежен. Альтернатива хорошему дизайну — плохой дизайн, а не отсутствие дизайна вообще.*
>
> *Book Design: A Practical Introduction by Douglas Martin*

Хотя комментарии Дугласа Мартина относятся не к проектированию программного обеспечения, они все же применимы к нашему ремеслу, где нет никакой альтернативы продуманному дизайну. В ситуации, описанной выше, если над проектом работают пять разработчиков, то отказ от проектирования не приведет к объединению пяти разных дизайнов в одном. Вы просто получите смесь пяти разных искусственных интерпретаций бизнес-языка без выгоды от привлечения реальных ЭКСПЕРТОВ ПРОБЛЕМНОЙ ОБЛАСТИ.

Подведем итог. Мы занимаемся моделированием независимо от того, признаем мы это или нет. Это можно сравнить с тем, как возникают и исчезают дороги. Некоторые дороги в далеком прошлом были оживленными транспортными магистралями, а потом постепенно зарастали и в конечном счете превращались в заброшенные тропы. Они делали необъяснимые повороты и разветвления, по которым ходили немногие путешественники. В какой-то момент эти дороги спрямили и замостили для удобства большего количества путешественников. В настоящее время по этим импровизированным дорогам почти никто не ходит, потому что они не были правильно спроектированы, а просто протаптывались наугад. Лишь немногие из наших современников догадываются, почему путешествие по ним является настолько неудобным и некомфортным. Современные дороги спланированы и спроектированы в результате продуманных исследований населения, окружающей среды и прогнозируемых транспортных потоков. Оба вида дорог были результатом моделирования. Однако в первом случае интеллект использовался в минимальной степени, а во втором — в максимальной. Программное обеспечение можно моделировать точно так же.

Если вы опасаетесь, что создание программного обеспечения на основе продуманного проектирования связано с большими расходами, подумайте, насколько дороже обойдется использование или даже исправление плохого дизайна. Это особенно относится к программному обеспечению, которое должно выделить вашу организацию среди всех других и принести вам значительное конкурентоспособное преимущество.

Со словом *хороший* тесно связано слово *эффективный*. Возможно, второе слово более точно выражает цель, к которой следует стремиться: *эффективный дизайн*. Он удовлетворяет потребности организации настолько, что позволяет ей получить конкурентное преимущество благодаря программному обеспечению. Стремление создать эффективный дизайн вынуждает организацию выяснить, какие аспекты обеспечивают ее превосходство и использовать их в качестве ориентира при разработке правильной модели программного обеспечения.

В методологии Scrum *приобретение знаний* осуществляется с помощью экспериментирования и совместного изучения предметной области и называется “приобретением информации” [Essential Scrum]. Знание никогда

не бывает бесплатным, но в этой книге я объясню, как ускорить его приобретение.

На всякий случай, если вы все еще сомневаетесь в том, что эффективный дизайн имеет большое значение, вспомните высказывания тех, кто, кажется, понимает его важность.

> *"Большинство людей делают ошибку, думая о дизайне, как о внешнем виде. Люди думают, что это видимость, что дизайнеру дают коробку и говорят: "Сделай так, чтобы она выглядела хорошо!» Это не то, что мы называем дизайном. Дизайн — это не просто внешний вид или восприятие вещи. Дизайн — это то, как она работает".*
>
> *Стив Джобс*

В программном обеспечении эффективный дизайн имеет огромное значение. Принимая во внимание альтернативу, я рекомендую эффективный дизайн.

Стратегическое проектирование

Мы начинаем со стратегического проектирования, важного во всех отношениях. Невозможно эффективно применять тактическое проектирование, если сначала не выполнить стратегическое проектирование. Стратегическое проектирование представляет собой картину, написанную крупными мазками, позволяющими углубиться в детали реализации. Он выделяет стратегически важные аспекты вашего бизнеса, показывает, как разделить работу по степени важности и как ее лучше всего объединить при необходимости.

Сначала вы узнаете, как выделить МОДЕЛЬ ПРЕДМЕТНОЙ ОБЛАСТИ (DOMAIN MODEL), используя шаблон стратегического проектирования под названием ОГРАНИЧЕННЫЙ КОНТЕКСТ (BOUNDED CONTEXT). Затем мы покажем, как раз-

работать ЕДИНЫЙ ЯЗЫК (UBIQUITOUS LANGUAGE) в качестве МОДЕЛИ ПРЕДМЕТНОЙ ОБЛАСТИ в пределах точно определенного ОГРАНИЧЕННОГО КОНТЕКСТА.

Вы узнаете, насколько важно привлекать к работе над ЕДИНЫМ ЯЗЫКОМ не только разработчиков, но и ЭКСПЕРТОВ ПРЕДМЕТНОЙ ОБЛАСТИ (DOMAIN EXPERTS). Вы увидите, как сотрудничают группы разработчиков программного обеспечения и ЭКСПЕРТЫ ПРЕДМЕТНОЙ ОБЛАСТИ. Это жизненно важное объединение умных и мотивированных людей, которое необходимо для того, чтобы предметно-ориентированное проектирование принесло наилучшие результаты. Язык, который вы разрабатываете, сотрудничая вместе, станет единым и вездесущим языком общения и моделирования.

По мере углубления в дебри стратегического проектирования вы столкнетесь с понятием ПОДОБЛАСТИ (SUBDOMAIN) и узнаете, как оно позволяет снизить сложность унаследованных систем и достичь отличных результатов при разработке совершенно новых проектов. Вы увидите также, как можно объединить многочисленные ОГРАНИЧЕННЫЕ КОНТЕКСТЫ с помощью методики под названием СВЯЗЫВАНИЕ КОНТЕКСТОВ (CONTEXT MAPPING). КАРТЫ КОНТЕКСТОВ (CONTEXT MAPS) определяют отношения между командой и техническими механизмами, существующими в двух объединенных ОГРАНИЧЕННЫХ КОНТЕКСТАХ.

Тактическое проектирование

После того как я дам вам прочные основы стратегического проектирования, вы обнаружите самые мощные средства тактического проектирова-

ния DDD. Тактическое проектирование можно сравнить с тонкой кисточкой, которая позволяет прорисовывать мелкие детали вашей модели предметной области. Одними из наиболее важных являются инструментальные средства, которые используются для агрегирования сущностей и объектов-значений в кластер правильного размера. Для этого предназначен шаблон АГРЕГАТ (AGGREGATE).

Предметно-ориентированное проектирование описывает моделирование вашей предметной области самым точным из возможных способов. Используя СОБЫТИЯ ПРЕДМЕТНОЙ ОБЛАСТИ (DOMAIN EVENTS), вы сможете точно моделировать предметную область и передавать информацию о том, что в ней происходит, в другие системы, которые должны знать об этом. Заинтересованные стороны могут находиться как в вашем ОГРАНИЧЕННОМ КОНТЕКСТЕ, так и в других отдаленных ОГРАНИЧЕННЫХ КОНТЕКСТАХ.

Процесс обучения и уточнения знаний

Подход DDD — это образ мышления, помогающий вам и вашей группе совершенствовать знания по мере изучения основных аспектов вашего бизнеса. В результате этого процесса возникают открытия, которые являются результатом обсуждений и экспериментов. Подвергая сомнению существующее положение вещей и бросая вызов вашим предположениям о

программной модели, ваша группа приобретет обширные и глубокие знания. Это крайне важные инвестиции в ваш бизнес и вашу группу. Цель состоит не только в обучении и совершенствовании, но и в максимально быстром обучении и совершенствовании. Для достижения этих целей существуют специальные инструменты, которые описываются в главе 7.

Поехали!

Даже в сжатом представлении информация о DDD является довольно обширной. Начнем с главы 2, "Стратегическое проектирование с помощью ОГРАНИЧЕННЫХ КОНТЕКСТОВ и ЕДИНОГО ЯЗЫКА".

Глава 2

Стратегическое проектирование с помощью ОГРАНИЧЕННЫХ КОНТЕКСТОВ и ЕДИНОГО ЯЗЫКА

Что такое ОГРАНИЧЕННЫЕ КОНТЕКСТЫ и ЕДИНЫЙ ЯЗЫК? В нескольких словах можно сказать, что предметно-ориентированное проектирование прежде всего подразумевает моделирование ЕДИНОГО ЯЗЫКА в четко определенном ОГРАНИЧЕННОМ КОНТЕКСТЕ. Несмотря на то что это определение совершенно правильное, все же, вероятно, это не самое полезное описание, которое я мог бы привести. Позвольте мне уточнить сказанное.

Во-первых, ОГРАНИЧЕННЫЙ КОНТЕКСТ — это семантическая контекстная граница. Это значит, что в пределах ОГРАНИЧЕННОГО КОНТЕКСТА каждый компонент программного обеспечения имеет определенное значение и делает определенные вещи. Компоненты внутри ОГРАНИЧЕННОГО КОНТЕКСТА являются характерными для данного контекста и семантически обоснованными. Это довольно просто.

Когда вы только приступаете к разработке своего программного обеспечения, ваш ОГРАНИЧЕННЫЙ КОНТЕКСТ носит преимущественно теоретический характер. Его можно интерпретировать как часть вашего *пространства задач* (problem space). Однако, по мере того как ваша модель становится все глубже и яснее, ваш ОГРАНИЧЕННЫЙ КОНТЕКСТ быстро переходит в *пространство решений* (solution space), а ваша программная модель отражается в исходном коде проекта. (Понятия пространства задач и пространства решений подробнее объясняются во врезке, приведенной ниже.) Помните, что ОГРАНИЧЕННЫЙ КОНТЕКСТ — это область, в которой реализуется модель, и каждому ОГРАНИЧЕННОМУ КОНТЕКСТУ соответствует отдельный артефакт программного обеспечения.

Пространства задач и решений

Пространство задач — это область, в которой осуществляется стратегический анализ высокого уровня и выполняются этапы проектирования с учетом заданных ограничений. Обсуждая главные факторы, влияющие на проект, и выделяя важные цели и риски, можно использовать простые диаграммы. На практике с большим успехом применяются КАРТЫ КОНТЕКСТОВ. Заметим также, что при необходимости в ходе обсуждения пространства задач можно использовать ОГРАНИЧЕННЫЕ КОНТЕКСТЫ, но они одновременно тесно связаны с пространством решений.

Пространство решений — это область, в которой фактически реализуется решение, идентифицированное в результате анализа пространства задач как СМЫСЛОВОЕ ЯДРО (CORE DOMAIN). СМЫСЛОВЫМ ЯДРОМ называется ОГРАНИЧЕННЫЙ КОНТЕКСТ, который разрабатывается как ключевая стратегическая инициатива организации. Решение в этом ОГРАНИЧЕННОМ КОНТЕКСТЕ преобразуется в исходный код — основной и тестовый. Кроме того, в пространстве решений разрабатывается код, поддерживающий интеграцию с другими ОГРАНИЧЕННЫМИ КОНТЕКСТАМИ.

Ограниченный контекст

Программная модель, функционирующая в ОГРАНИЧЕННОМ КОНТЕКСТЕ, отражает язык, на котором говорят все ее разработчики. Этот язык называется ЕДИНЫМ ЯЗЫКОМ, потому что он используется как для общения в группе разработчиков, так и для реализации модели. Таким образом, ЕДИНЫЙ ЯЗЫК должен быть формальным — строгим, точным, конкретным и выразительным. На диаграмме прямоугольники внутри ОГРАНИЧЕННОГО КОНТЕКСТА представляют концепции модели, которые могут быть реализованы в виде классов. Если ОГРАНИЧЕННЫЙ КОНТЕКСТ разрабатывается как ключевая стратегическая инициатива организации, она называется СМЫСЛОВЫМ ЯДРОМ.

По сравнению со всем программным обеспечением, которое использует ваша организация, СМЫСЛОВОЕ ЯДРО представляет собой самую важную программную модель, потому что именно оно должно принести организации конкурентное преимущество. СМЫСЛОВОЕ ЯДРО разрабатывается так, чтобы ваша организация могла конкурировать со всеми остальными компаниями. Оно должно решать хотя бы основные проблемы вашего бизнеса. Ваша организация не может превосходить всех и во всем, можете даже не пытаться. Следовательно, необходимо тщательно обдумать, каким должно быть СМЫСЛОВОЕ ЯДРО и каким оно не должно быть. Это первое ценностное предложение DDD, позволяющее организации правильно выбрать СМЫСЛОВОЕ ЯДРО и выделить для него лучшие ресурсы.

Когда какой-нибудь член группы использует выражение ЕДИНОГО ЯЗЫКА, все остальные члены группы должны совершенно точно понимать его смысл. Это выражение должно быть широко распространенным в пределах группы, как и весь язык, определяющий разрабатываемую программную модель.

Размышляя о языке программной модели, вспомните о разных народах, населяющих Европу. В каждой из европейских стран существует официальный язык, на котором говорят все ее жители. Например, в Германии, Франции и Италии официальные языки определены точно. Когда вы пересекаете границу, официальный язык изменяется. То же самое касается Азии, где в Японии говорят на японском языке, а языки, на которых говорят в Китае и Корее, четко ограничены национальными границами. ОГРАНИЧЕННЫЕ КОНТЕКСТЫ можно сравнить с языковыми границами. В случае DDD речь идет о языках, на которых говорят члены группы разработчиков программной модели, а также о языках программирования, на которых написан ее исходный код.

Ограниченные контексты, группы и репозитории исходных текстов

Над каждым ОГРАНИЧЕННЫМ КОНТЕКСТОМ должна работать только одна группа. Кроме того, каждый ОГРАНИЧЕННЫЙ КОНТЕКСТ должен иметь отдельный репозиторий исходного кода. Одна группа может работать над несколькими ОГРАНИЧЕННЫМИ КОНТЕКСТАМИ, но над одним ОГРАНИЧЕННЫМ

КОНТЕКСТОМ не должны работать несколько групп. Четко отделите исходный код и схему базы данных каждого ОГРАНИЧЕННОГО КОНТЕКСТА так же, как вы отделяете ЕДИНЫЙ ЯЗЫК контекста от остальных контекстов. Храните приемочные тесты и тестовые модули вместе с основными исходными кодами.

Еще раз подчеркнем очень важное требование, что одна группа должна работать только над одним ОГРАНИЧЕННЫМ КОНТЕКСТОМ. Это полностью устраняет возможности любых неприятных сюрпризов, которые могут возникнуть, когда другая группа вносит изменения в ваш исходный код. Ваша группа имеет свой исходный код и базу данных и определяет официальные интерфейсы, с помощью которых должен использоваться ваш ОГРАНИЧЕННЫЙ КОНТЕКСТ. В этом заключается одно из преимуществ использования DDD.

Ограниченный контекст

В естественных языках терминология развивается в течение долгого времени, и в разных странах одни и те же или похожие слова имеют разные оттенки. Вспомните, например, о различиях между словами испанского языка в Испании и Колумбии, где даже произношение изменилось. Очевидно, что существуют испанский язык Испании и испанский язык Колумбии. Это же можно сказать о языках программных моделей. Вполне возможно, что люди в разных группах приписывают одному и тому же термину разные значения, потому что их бизнес-знания относятся к разным контекстам; они разрабатывают разные ОГРАНИЧЕННЫЕ КОНТЕКСТЫ. Любые компоненты вне контекста могут не описываться теми же определениями. Скорее всего, они будут в той или иной степени отличаться от компонентов, которые моделируются вашей группой. Это прекрасно.

Для того чтобы понять одну весомую причину использования ОГРАНИЧЕННЫХ КОНТЕКСТОВ, рассмотрим типичную проблему проектирования программного обеспечения. Часто группы не знают, когда следует прекратить накапливать все новые и новые концепции в своей модели предметной области. В начале модель может быть маленькой и управляемой...

Но затем группа добавляет в модель все больше и больше концепций. Вскоре это приводит к большой проблеме. Мало того, что появляется слишком много концепций, но и язык модели становится нечетким, потому что, когда вы думаете о нем, то на самом деле используете несколько языков в одной большой и запутанной модели.

Из-за этой ошибки группы часто превращают новое программное изделие в то, что называют БОЛЬШИМ КОМОМ ГРЯЗИ (BIG BALL OF MUD). Безусловно, БОЛЬШОЙ КОМ ГРЯЗИ не может быть предметом гордости. Это монолит, и даже еще хуже. Он представляет собой систему, состоящую из многочисленных запутанных моделей без четких границ. Для работы над ним потребуется несколько разных команд, что очень проблематично. Кроме того, разные независимые концепции разбросаны по многочисленным модулям модулей и взаимосвязаны с конфликтующими элементами. Если в таком проекте предусмотрено тестирование, то оно может занять очень много времени, и поэтому тестами могут пренебречь именно тогда, когда они на самом деле необходимы.

БОЛЬШОЙ КОМ ГРЯЗИ — это результат попытки сделать слишком много, со слишком многими людьми и в неправильном месте. Любая попытка разработать и использовать ЕДИНЫЙ ЯЗЫК приведет к ломаному и плохо определенному диалекту, от которого вскоре все будут вынуждены отказаться. Этот язык не будет похож даже на эсперанто. Это просто смесь, как БОЛЬШОЙ КОМ ГРЯЗИ.

Эксперты проблемной области и бизнес-факторы

Иногда прямые подсказки или тонкие намеки бизнес-партнеров позволяют техническим специалистам принимать более правильные решения. Таким образом, БОЛЬШОЙ КОМ ГРЯЗИ часто является результатом стихийных усилий, предпринятых группой разработчиков программного обеспечения, которые не слушают бизнес-экспертов.

Подсказать, где проходят границы модели, может разделение организации на отделы или рабочие группы. Обычно всегда можно найти хотя бы одного бизнес-эксперта по конкретной бизнес-функции. В последнее время наблюдается тенденция объединять людей, работающих над проектом, в группы, а подразделения и даже функциональные группы, образующие иерархию управления, становятся все менее популярными. Даже столкнувшись с новыми бизнес-моделями, вы все еще будете обнаруживать, что проекты организованы в соответствии с бизнес-факторами и в границах областей знания. Возможно, о разделении функций следует думать именно в этих терминах.

Такое разделение необходимо, когда анализ показывает, что каждая бизнес-функция может использовать разные определения одного и того же термина. Разберем концепцию *политика* и посмотрим, как изменяется ее значение в разных областях страхового дела. Легко понять, что термин *политика* при подписании страхового договора совершенно отличается от политики урегулирования претензий или инспектирования. Более подробно эти различия описаны во врезке, приведенной ниже.

Термин *политика* в каждой из этих областей имеет разный смысл. Этот факт нельзя игнорировать или изменить путем умственных упражнений.

Различия в толковании термина *политика*

Политика заключения страхового договора. В области знаний, связанной с заключением страховых договоров, политика означает правила оценки рисков, которым подвергается предмет страхования. Например, работая над договором страхования, страховщики должны оценить риски, связанные с активами, чтобы вычислить размер страховой премии по договору и правильно компенсировать убытки по данным активам.

Политика инспектирования. Страховые компании всегда имеют подразделение инспекторов, проверяющих состояние предметов страхования. Страховщики в некоторой степени зависят от информации, обнаруженной в ходе проверок, но только с той точки зрения, что состояние предмета страхования должно соответствовать условиям, указанным в договоре страхования. Если условия договора выполнены, то детали проверки — фотографии и описания — относятся к политике инспектирования. На эти данные можно ссылаться при подписании договора и определении премиальной стоимости.

Политика урегулирования претензий. Политика урегулирования претензий регламентирует процесс рассмотрения претензии на возмещение убытков в терминах политики заключения страховых договоров. Политика урегулирования претензий должна ссылаться на правила заключения страховых договоров, но в основном она сфокусирована на оценке повреждений предмета страхования и результатах осмотров, выполненных персоналом в соответствии с процедурами, чтобы определить размер выплат, если их вообще следует делать.

Если вы попробуете объединить все три вида политики в одну политику для всех трех бизнес-групп, то, конечно, столкнетесь с проблемами. Это стало бы еще более проблематичным, если термин *политика* впоследствии будет перегружен четвертым и пятым толкованиями. Все окажутся в проигрыше.

С другой стороны, DDD подчеркивает такие различия с помощью разделения разных типов по разным ОГРАНИЧЕННЫМ КОНТЕКСТАМ. При этом считается, что каждый контекст имеет свой язык и соответственно функционирует. Существуют три толкования термина *политика*? Значит, существуют три ОГРАНИЧЕННЫХ КОНТЕКСТА, каждый со своей собственной политикой, имеющей уникальные особенности. Нет никакой потребности называть их `UnderwritingPolicy`, `ClaimsPolicy` или `InspectionsPolicy`. Область их видимости определяется именем соответствующего ОГРАНИЧЕННОГО КОНТЕКСТА. Назовите эту концепцию просто `Policy` во всех трех ОГРАНИЧЕННЫХ КОНТЕКСТАХ.

Другой пример: что такое полет?

В отрасли авиаперевозок термин *полет* (flight) может иметь несколько значений. Есть полет, который определен как взлет и приземление, в результате которых самолет перемещается из одного аэропорта в другой. Есть другой вид полета, определяемый в терминах, связанных с обслуживанием самолета. И есть еще один полет, определенный в терминах, связанных с покупкой билетов пассажирами, — беспосадочный или с пересадкой. Поскольку каждое из этих толкований термина *полет* становится ясным только в его контексте, каждое из них должно моделироваться в отдельном ОГРАНИЧЕННОМ КОНТЕКСТЕ. Моделирование всех трех терминов в одном и том же ОГРАНИЧЕННОМ КОНТЕКСТЕ привело бы к путанице.

Типичный пример

Для того чтобы полнее разъяснить причины использования ОГРАНИЧЕННЫХ КОНТЕКСТОВ, позвольте мне проиллюстрировать их на примере одной предметной области. Представим себе проект, разрабатываемый на основе методологии гибкого проектирования Scrum. Его основным понятием является Продукт (`Product`)[1] — программное обеспечение, которое должно быть создано и впоследствии усовершенствовано. Понятие Продукт подразделяется на Элементы бэклога (`BacklogItem`), Выпуски (`Release`) и Спринты (`Sprint`). Каждый Элемент бэклога состоит из множества Задач (`Task`), каждая из которых может иметь коллекцию Записей журнала оценок (`EstimationLogEntry`). Выпуски содержат Запланированные элементы журнала невыполненных работ (`ScheduledBacklogItems`), а Спринты — Назначенные

[1] В скобках указаны имена соответствующих классов, приведенных на диаграмме. — Примеч. ред.

элементы бэклога (`CommittedBacklogItem`). Пока неплохо. Мы идентифицировали основные понятия нашей модели предметной области, а язык точен и прозрачен.

"О, да — говорят члены группы, — давайте не забывать о наших пользователях. Мы хотим облегчить совместные обсуждения в группе разработчиков продукта. Давайте представлять каждую организацию-подписчика как Арендатора (`Tenant`). Внутри Арендатора мы позволим регистрацию любого количества Пользователей (`User`), которые будут иметь Полномочия (`Permission`). Кроме того, давайте добавим понятие Обсуждение (`Discussion`), чтобы представить инструменты взаимодействия, которые мы поддерживаем".

Затем члены группы добавляют: "Хорошо, но существуют и другие инструменты взаимодействия. Обсуждения можно вести на Форумах (`Forum`), кроме того, Обсуждения связаны с Сообщениями (`Post`). Также мы хотим поддержать Общие календари (`SharedCalendar`)".

Они продолжают: "И не забывайте, что нам нужен способ, которым Арендаторы будут делать Платежи (`Payment`). Мы также предлагаем гибкие планы поддержки, поэтому нам нужен способ слежения за процессом поддержки. И Поддержка, и Платежи должны управляться Учетной записью (`Account`)".

Со временем появляется все больше концепций: "Каждый Продукт, разрабатываемый на основе методологии Scrum, имеет конкретную Группу (`Team`), которая воздействует на Продукт. Группы состоят из одного Владельца Продукта (`ProductOwner`) и множества Членов Группы (`TeamMember`). Но как учесть вопросы, связанные с использованием человеческих ресурсов? Что, если мы смоделируем Графики работы членов группы (`Schedule`) вместе с эффективностью их работы и доступностью?"

"Знаете что? — говорят члены группы. — Общие Календари не должны ограничиваться пустыми Календарными записями (`CalendarEntry`). Мы должны иметь возможность идентифицировать конкретные виды Календарных записей вроде Напоминаний (`Reminder`), Контрольных сроков (`Milestone`), Запланированных встреч (`Planning`) и Состоявшихся встреч (`Retrospective`), а также Расчетных сроков (`TargetDate`)".

Остановитесь на минуту! Вы видите западню, в которую попала эта группа? Посмотрите, как далеко они отклонились от первичных основных концепций Продукт, Элементы бэклога, Выпуски и Спринты. Язык больше не описывает только методологию Scrum; он стал запутанным и непонятным.

Не заблуждайтесь относительно небольшого количества именованных концепций. За каждым именованным элементом скрываются две или три связанных с ними концепции, которые немедленно всплывают в памяти. Группа уже далеко продвинулась к поставке БОЛЬШОГО КОМА ГРЯЗИ, хотя проект еще только начался.

Необходимость фундаментального стратегического проектирования

Какие инструменты DDD позволяют избежать описанных выше ловушек? Для этого нужны по крайней мере два инструмента фундаментального стратегического проектирования. Первый — ОГРАНИЧЕННЫЙ КОНТЕКСТ, второй — ЕДИНЫЙ ЯЗЫК. Использование ОГРАНИЧЕННОГО КОНТЕКСТА заставляет нас ответить на вопрос: "Что является ядром?" ОГРАНИЧЕННЫЙ КОНТЕКСТ должен объединять близкие концепции, которые являются ядром стратегической инициативы, и отбросить все остальное. Оставшиеся концепции являются частью ЕДИНОГО ЯЗЫКА группы. В дальнейшем мы покажем, как подход DDD позволяет избежать создания монолитных приложений.

Проверка преимуществ

Поскольку ОГРАНИЧЕННЫЕ КОНТЕКСТЫ не монолитны, их использование приносит дополнительные преимущества. Одно из них состоит в том, что тесты сосредоточиваются на одной модели, а значит, их количество уменьшается и они выполняются быстрее. Хотя это не главное преимущество ОГРАНИЧЕННЫХ КОНТЕКСТОВ, оно окупается во многих ситуациях.

Говоря буквально, некоторые концепции окажутся в контексте и будут явно включены в язык группы.

Другие концепции будут находиться вне контекста. Концепции, которые пройдут строгий отбор для включения в ядро, станут частью ЕДИНОГО ЯЗЫКА группы, которая владеет ОГРАНИЧЕННЫМ КОНТЕКСТОМ.

Примите к сведению

Концепции, прошедшие строгий отбор для включения в ядро, являются частью ЕДИНОГО ЯЗЫКА группы, владеющей ОГРАНИЧЕННЫМ КОНТЕКСТОМ. Граница подчеркивает строгий порядок внутри контекста.

Разработчики

Эксперты предметной области

Так что же такое ядро? Это область, в которой мы должны объединить две группы людей в единую команду соратников — экспертов предметной области и разработчиков программного обеспечения.

Мысли ЭКСПЕРТОВ ПРЕДМЕТНОЙ ОБЛАСТИ, естественно, будут больше сосредоточены на вопросах бизнеса: в центре внимания будет их представление о том, как работает их бизнес. В методологии Scrum ЭКСПЕРТ ПРЕДМЕТНОЙ ОБЛАСТИ играет роль Scrum-мастера, который полностью понимает, как Scrum применяется к проекту.

ВЛАДЕЛЕЦ ПРОДУКТА ИЛИ ЭКСПЕРТ ПРЕДМЕТНОЙ ОБЛАСТИ?

Вы можете задаться вопросом, в чем различие между ВЛАДЕЛЬЦЕМ ПРОДУКТА Scrum и ЭКСПЕРТОМ ПРЕДМЕТНОЙ ОБЛАСТИ DDD. В некоторых случаях эти роли может играть один и тот же человек, способный их выполнить. И все же не следует удивляться тому, что ВЛАДЕЛЕЦ ПРОДУКТА обычно сильнее сосредоточивается на управлении и распределении приоритетов в бэклоге продукта, стремясь поддерживать концептуальную и техническую целостность проекта. Однако это не означает, что ВЛАДЕЛЕЦ ПРОДУКТА естественным образом является ЭКСПЕРТОМ ПРЕДМЕТНОЙ ОБЛАСТИ, над которой вы работаете. Включайте в группу только настоящих ЭКСПЕР-

ТОВ ПРЕДМЕТНОЙ ОБЛАСТИ и не заменяете их ВЛАДЕЛЬЦАМИ ПРОДУКТА, не владеющими необходимым ноу-хау.

В вашем конкретном бизнесе также существуют ЭКСПЕРТЫ ПРЕДМЕТНОЙ ОБЛАСТИ. Это не должность, а характеристика людей, которых главным образом интересует предметная область. Именно их ментальная модель является отправной точкой для формирования ЕДИНОГО ЯЗЫКА группы.

С другой стороны, разработчики сосредоточены на разработке программного обеспечения. Как показано на рисунке, разработчики могут быть поглощены языками программирования и технологиями. И все же разработчики, работающие над проектом в рамках подхода DDD, должны упорно сопротивляться соблазну зациклиться только на технических вопросах и не вникать в деловой смысл основной стратегической инициативы. Разработчики должны отклонять любую неуместную лаконичность и быть в состоянии освоить ЕДИНЫЙ ЯЗЫК, который постепенно вырабатывает их группа в конкретном ОГРАНИЧЕННОМ КОНТЕКСТЕ.

Сосредоточьтесь на сложности предметной области, а не на технической сложности

Вы используете DDD, потому что сложность предметной области высока. Мы никогда не хотим делать модель предметной области более сложной, чем она должна быть. Однако вы используете DDD, потому что модель предметной области сложнее, чем технические аспекты проекта. Именно поэтому разработчики должны вникнуть в модель предметной области вместе с ЭКСПЕРТАМИ ПРЕДМЕТНОЙ ОБЛАСТИ!

И разработчики, и ЭКСПЕРТЫ ПРЕДМЕТНОЙ ОБЛАСТИ не должны допустить, чтобы документы заменили живое общение. Лучший ЕДИНЫЙ ЯЗЫК можно выработать только при наличии обратной связи, управляющей объединенной ментальной моделью группы. Открытый разговор, исследование и обогащение ваших знаний будут способствовать более глубокому пониманию СМЫСЛОВОГО ЯДРА.

Ставьте проблемы и обобщайте

Вернемся к вопросу: что является ядром? Используя прежде неуправляемую и постоянно расширяющуюся модель, ставьте проблемы и обобщайте!

Есть одна очень простая проблема — выяснить, соответствует ли каждая из концепций крупной модели ЕДИНОМУ ЯЗЫКУ Scrum. Например, концепции Арендатор (`Tenant`), Пользователь (`User`) и Полномочия (`Permission`) не имеют ничего общего с методологией Scrum. Эти концепции должны быть исключены из нашего программного обеспечения.

Концепции Арендатор, Пользователь и Полномочия (`Permission`) должны быть заменены концепциями Группа (`Team`), Владелец продукта (`ProductOwner`) и Член группы (`TeamMember`). Концепции Владелец продукта и Член группы фактически представляют собой концепцию Пользователь внутри концепции Аренда (`Tenancy`), но Владелец продукта и Член группы — это термины ЕДИНОГО ЯЗЫКА Scrum. Это естественные термины, которые мы используем, говоря о Scrum-продуктах и работе группы, которая их разрабатывает.

Являются ли Планы поддержки (`SupportPlan`) и Платежи (`Payment`) частью управления проектами Scrum? Ответ очевиден — нет. Правда, Планы поддержки и Платежи будут управляться Учетной записью (`Account`) Арендатора, но они не являются частью основного языка Scrum. Они находятся вне контекста и должны быть удалены из модели.

Не следует ли нам ввести концепции управления человеческими ресурсами? Эти концепции, вероятно, для кого-нибудь могут оказаться полезными, но они не будут непосредственно использоваться Волонтерами (`Volunteer`), представляющими собой разновидность Членов группы (`TeamMember`), которые будут работать над Задачами, связанными с элементами бэклога (`BacklogItemTasks`). Они находятся вне контекста.

После добавления концепций Группа, Владелец продукта и Член группы разработчики модели поняли, что они пропустили основную концепцию, позволяющую Членам группы работать над Задачами. В методологии Scrum эта концепция называется Волонтер (`Volunteer`). Следовательно, концепция Волонтер находится в контексте и должна быть включена в язык основной модели.

Несмотря на то что концепции Контрольный срок (`Milestone`), Состоявшаяся встреча (`Retrospective`) и им подобные находятся в контексте, группа предпочла бы отложить их до более поздних спринтов. Они находятся в контексте, но пока отсутствуют в области видимости.

Наконец, разработчики модели хотят удостовериться, что текущие **Обсуждения** (`Discussion`) будут частью основной модели. В результате они моделируют **Обсуждение**. Это означает, что **Обсуждение** является частью ЕДИНОГО ЯЗЫКА группы, а значит, находится в ОГРАНИЧЕННОМ КОНТЕКСТЕ.

Эти лингвистические проблемы сделали основную модель и модель ЕДИНОГО ЯЗЫКА более ясными. И все же, как будет модель Scrum осуществлять необходимые **Обсуждения**? Для этого, конечно, потребуется большое количество вспомогательного программного обеспечения. Таким образом, моделировать **Обсуждения** в ОГРАНИЧЕННОМ КОНТЕКСТЕ Scrum нецелесообразно. Фактически весь набор концепций **Сотрудничество** находится вне контекста. **Обсуждение** будет поддерживаться путем интегрирования с другим ОГРАНИЧЕННЫМ КОНТЕКСТОМ — **Контекстом Сотрудничества**.

После такого отбора концепций фактическое СМЫСЛОВОЕ ЯДРО становится намного меньше. Конечно, оно будет расти. Мы уже знаем, что со временем в него будут включены Контрольные сроки, Состоявшиеся встречи и модели календаря. Однако модель будет увеличиваться только за счет новых концепций, придерживающихся ЕДИНОГО ЯЗЫКА Scrum.

А что можно сказать о всех других концепциях моделирования, которые были удалены из СМЫСЛОВОГО ЯДРА? Вполне возможно, что некоторые из этих концепций, если не все, образуют свои собственные ОГРАНИЧЕННЫЕ

КОНТЕКСТЫ, в каждом из которых будет свой ЕДИНЫЙ ЯЗЫК. Позже вы увидите, как мы интегрируем их с помощью КАРТ КОНТЕКСТОВ.

Разработка ЕДИНОГО ЯЗЫКА

Итак, как же фактически разработать ЕДИНЫЙ ЯЗЫК вашей группы, раз вы решили внедрить в практику этот мощный инструмент предметно-ориентированного проектирования? Должен ли он состоять из хорошо известных имен существительных? Существительные важны, но часто разработчики программ придают слишком много внимания существительным в рамках модели предметной области, забывая, что разговорный язык состоит не только из них. Правда, в предыдущем ОГРАНИЧЕННОМ КОНТЕКСТЕ мы главным образом сосредоточились именно на существительных, но это объяснялось тем, что нас интересовал другой аспект DDD, который позволяет выделить элементы СМЫСЛОВОГО ЯДРА.

Методы ускорения работы

Работая над сценариями, вы можете провести несколько сеансов СОБЫТИЙНОГО ШТУРМА. Они помогут вам быстро понять, над какими сценариями следует работать, и правильно расставить их приоритеты. Аналогично разработка конкретных сценариев позволит вам выбрать оптимальное направление сеансов СОБЫТИЙНОГО ШТУРМА. Эти два инструмента хорошо работают вместе. СОБЫТИЙНЫЕ ШТУРМЫ описываются в главе 7.

Не ограничивайте ваше СМЫСЛОВОЕ ЯДРО одними существительными. Вместо этого представьте СМЫСЛОВОЕ ЯДРО как ряд конкретных сценариев, которые должна реализовывать модель предметной области. Когда я говорю "сценарии", я не имею в виду прецеденты или пользовательские истории, типичные для программных проектов. Я имею в виду буквальный смысл слова "сценарий", т.е. как модель предметной области должна работать и что именно должны делать ее компоненты. Этой цели можно достичь только в результате сотрудничества ЭКСПЕРТОВ ПРЕДМЕТНОЙ ОБЛАСТИ и разработчиков.

Ниже приведен пример сценария, соответствующего ЕДИНОМУ ЯЗЫКУ Scrum.

Разрешить назначение любого элемента бэклога для спринта. Элемент бэклога может быть назначен для спринта, только если он уже намечен для выпуска. Если он ранее был назначен для другого спринта, то это назначение необходимо сначала отменить. После того как эле-

мент бэклога будет назначен для спринта, известите об этом заинтересованные стороны.

Обратите внимание на то, что это не просто сценарий, описывающий то, как люди используют Scrum для работы над проектом. Мы не говорим о процедурах с участием людей. Скорее этот сценарий представляет собой описание того, как вполне реальные компоненты модели программного обеспечения используются для организации управления проектом на основе методологии Scrum.

Предыдущий сценарий не совсем четко сформулирован, и преимущество DDD состоит в том, что мы постоянно находимся в поиске способов улучшить модель. И все же это вполне приличное начало. Мы видим имена существительные, но наш сценарий не ограничивается только ими, в нем также есть глаголы, наречия и другие грамматические конструкции. Мы также видим ограничения — условия, которые должны быть выполнены, чтобы сценарий был успешно завершен. Самое важное преимущество и характерная особенность DDD заключаются в том, что вы реально можете обсуждать работу модели предметной области, т.е. ее дизайн.

Можно даже рисовать простые изображения и диаграммы. Этого достаточно, чтобы члены группы хорошо понимали друг друга. Однако здесь уместно сделать одно предупреждение. Внимательно следите за тем, чтобы время, потраченное на моделирование предметной области, т.е. разработку сценариев и рисование диаграмм, не было слишком долгим. Эти вопросы не относятся к модели предметной области. Это лишь вспомогательные средства, помогающие разрабатывать модель предметной области. В конце концов, код — это модель, а модель — это код. Церемонии, например свадьбы, предназначены для соблюдения обычаев, а не для моделирования предметной области. Это не означает, что вы должны воздерживаться от любых попыток обновить сценарии, но только делать это надо не слишком долго, чтобы стремление к совершенству не мешало работе.

Что следует сделать, чтобы улучшить часть ЕДИНОГО ЯЗЫКА в предыдущем примере? Задумайтесь об этом на минуту. Что мы пропустили? Скорее всего, вы захотите узнать, *кто* назначает элемент бэклога для спринта. Давайте укажем *кто* и посмотрим, что получится.

Владелец продукта назначает каждый элемент бэклога для спринта...

Во многих случаях вы обнаружите, что должны назвать каждую персону, участвующую в сценарии, и приписать другим концепциям вроде элемента бэклога и спринта их отличительные атрибуты. Это поможет конкретизировать ваш сценарий, и он станет меньше похож на перечисление критериев приемлемости. Однако в данном конкретном случае нет весомых при-

чин называть владельца продукта или далее описывать элемент бэклога и соответствующий спринт. В этом случае все владельцы продукта, элементы бэклога и спринты будут работать одинаково, независимо от того, конкретные они или общие. Если же в каких-то ситуациях вам придется назвать имена и перечислить атрибуты концепций, то используйте их.

Владелец продукта Изабель назначает элемент бэклога Представление профиля пользователя для спринта Поставка профилей пользователей...

Теперь сделаем небольшую паузу. Владелец продукта — не единственный человек, на которого возложена ответственность за включение элемента бэклога в спринт. Группы, применяющие методологию Scrum, не очень любят такие ситуации, потому что тогда они вынуждены поставлять программное обеспечение в течение установленного периода времени, не имея права голоса в его определении. Впрочем, в нашей программной модели может оказаться самым практичным решением возложить ответственность за данное конкретное действие над моделью на конкретного человека. В таком случае мы утверждаем, что он играет роль владельца продукта. Но даже в этом случае природа методологии Scrum порождает вопрос: "Не должны ли остальные члены группы сделать что-либо, чтобы помочь владельцу продукта выполнить свои обязанности?"

Вы видите, что случилось? Поставив вопрос "*Кто?*", мы стали глубже разбираться в модели. Возможно, мы должны потребовать, по крайней мере, существования консенсуса в группе о том, что элемент бэклога может быть назначен для спринта до фактического разрешения владельца продукта выполнить эту операцию. Это может привести к следующему усовершенствованному сценарию.

Владелец продукта назначает элемент бэклога в спринт. Элемент бэклога может быть назначен для спринта, только если он уже намечен для выпуска и если кворум членов группы одобрил эту операцию...

Отлично, теперь мы уточнили ЕДИНЫЙ ЯЗЫК, потому что идентифицировали новую концепцию модели — **Кворум**. Мы решили, что в группе должен быть **Кворум** ее членов, которые соглашаются с тем, что элемент бэклога должен быть назначен для спринта, и у них должен быть способ *одобрить* это обещание. В результате появилась новая концепция моделирования и некоторая идея о том, что пользовательский интерфейс должен облегчать взаимодействие в группе. Вы видите, как разворачиваются инновации?

В нашей модели пропущен еще один ответ на вопрос "*Кто?*" Какой? Посмотрим на завершение нашего сценария.

После того как элемент бэклога будет назначен для спринта, известите об этом заинтересованные стороны.

Кто или что является заинтересованной стороной? Этот вопрос и ответ на него ведет к моделированию точек зрения. Кто должен знать, когда элемент бэклога назначен для спринта? Конечно, сам спринт, представляющий собой важный элемент модели. Спринт должен отслеживать все назначения и усилия, требующиеся для поставки всех задач спринта. Если вы решите спроектировать спринт так, чтобы обеспечить такое слежение, то обязаны уведомить спринт о каждом назначении элементов бэклога.

Если он ранее был назначен для другого спринта, то сначала необходимо отменить это назначение. После того как элемент бэклога будет назначен для спринта, известите об этом спринт, назначение для которого было отменено, и спринт, для которого назначен данный элемент.

Теперь у нас получился довольно приличный сценарий предметной области. Заключительное предложение также привело нас к пониманию, что элемент бэклога и спринт могут узнать о назначении не одновременно. Бизнес должен быть точным, хотя это больше похоже на общепринятое требование *итоговой согласованности* (eventual consistency). Почему это важно и как этого достичь, объясняется в главе 5.

Полный вариант усовершенствованного сценария выглядит следующим образом.

Владелец продукта назначает элемент бэклога для спринта. Элемент бэклога может быть назначен для спринта, только если он уже намечен для выпуска и если кворум членов группы одобрил эту операцию. Если он ранее был назначен для другого спринта, то это назначение необходимо сначала отменить. После того как элемент бэклога будет назначен для спринта, известите об этом спринт, назначение для которого было отменено, и спринт, для которого назначен данный элемент.

Как программная модель работает на практике? Легко представить себе весьма инновационный пользовательский интерфейс, поддерживающий эту модель программного обеспечения. Когда группа Scrum участвует в сеансе планирования спринта, члены группы используют свои смартфоны или другие мобильные устройства, чтобы высказать свое одобрение каждого элемента бэклога в ходе обсуждения и согласования работ в рамках следующего спринта. Консенсус кворума членов группы, одобряющих каждый из элементов бэклога, дают владельцу продукта возможность назначать все одобренные элементы бэклога для спринта.

Реализация сценариев

Вы можете задаться вопросом, как перейти от письменного сценария к некоему артефакту, который можно использовать для проверки вашей модели предметной области на соответствие спецификациям, разработанным группой. Для этого можно использовать методику под названием *спецификация на примерах* (Specification by Example) [Specification]; ее также называют *разработкой, основанной на поведении* (Behavior-Driven Development) [BDD]. С помощью этого подхода вы можете обеспечить совместную разработку и уточнение ЕДИНОГО ЯЗЫКА, моделирование на основе общих принципов и проверку соответствия вашей модели вашим спецификациям. Для этого организуются приемочные тесты. Ниже показано, как можно переписать предыдущий сценарий в виде исполняемой спецификации.

```
Сценарий: владелец продукта назначает элемент бэклога для спринта
 Если данный элемент бэклога намечен для выпуска
 И владелец продукта, к которому относится элемент бэклога
 И целевой спринт
 И кворум членов группы одобряют это назначение
 Когда владелец продукта назначает элемент бэклога для спринта
 Тогда элемент бэклога назначается для спринта
 И генерируется событие назначения элемента бэклога для спринта
```

На основе сценария, написанного в таком виде, можно написать код и использовать какой-нибудь инструмент, чтобы выполнить эту спецификацию. Даже не имея такого инструмента, вы можете прийти к выводу, что эта форма сценария на основе подхода если/когда/тогда, работает лучше, чем его предыдущий вариант. И все же выполнение спецификаций для проверки соответствия модели предметной области может оказаться очень соблазнительным. Я прокомментирую это в главе 7.

Вы не должны использовать эту форму выполняемой спецификации для проверки вашей модели предметной области по вашим сценариям. Для этого лучше использовать принципы модульного тестирования, на основе которых создаются приемочные тесты (но не модульные), которые проверяют вашу модель предметной области.

```
/*
Владелец продукта включает элемент бэклога в спринт. Элемент бэклога
может быть включен в спринт, только если он уже намечен для выпуска,
и если кворум членов группы одобрил эту операцию. После того как
элемент бэклога будет назначен для спринта, известите об этом спринт,
для которого назначен данный элемент.
*/
```

```
[Тест]
public void ShouldCommitBacklogItemToSprint()
{
 // Если
 var backlogItem = BacklogItemScheduledForRelease();

 var productOwner = ProductOwnerOf (backlogItem);

 var sprint = SprintForCommitment ();

 var quorum = QuorumOfTeamApproval (backlogItem, sprint);

 // Когда
 backlogItem. CommitTo (sprint, productOwner, quorum);

 // Тогда
 Assert. IsTrue (backlogItem. IsCommitted());

 var backlogItemCommitted =
 backlogItem.Events.OfType<BacklogItemCommitted>().SingleOrDefault ();

 Assert.IsNotNull (backlogItemCommitted);
}
```

Этот подход к приемочным испытаниям на основе модульных тестов достигает той же цели, что и выполняемая спецификация. Его преимущество заключается в возможности быстрее проверять правильность сценария, правда, за счет некоторого ухудшения читабельности кода. Однако большинство ЭКСПЕРТОВ ПРЕДМЕТНОЙ ОЛБАСТИ должно быть в состоянии читать этот код с определенными пояснениями его разработчика. Этот подход, вероятно, работает лучше всего, если код содержит словесное описание сценария, как в данном примере.

Какой бы подход к тестированию вы ни выбрали, оба они будут использоваться в режиме семафора (зеленый цвет — успех, красный —провал), когда ваша спецификация сначала будет терпеть неудачу при выполнении, потому что у вас пока нет никаких концепций модели предметной области, которые можно было бы проверить. Шаг за шагом уточняя модель предметной области на основе ряда провальных результатов тестирования (которые выделяются красным цветом), вы получите полностью выверенную спецификацию, которая проходит все тесты (все будет закрашено зеленым цветом). Эти приемочные испытания будут непосредственно связаны с вашим ОГРАНИЧЕННЫМ КОНТЕКСТОМ и храниться в репозитории исходных текстов.

Далекие перспективы

Теперь у вас может возникнуть вопрос: как поддерживать ЕДИНЫЙ ЯЗЫК, когда закончится период внедрения и начнется период обслуживания? На практике приобретение новых знаний происходит в течение долгого времени, даже на протяжении периода обслуживания. Многие группы делают типичную ошибку, прекращая совершенствовать язык, когда начинается обслуживание.

Возможно, худшее, что можно сделать, — это распространить концепцию "стадия обслуживания" на СМЫСЛОВОЕ ЯДРО. Процесс непрерывного изучения в принципе не может ограничиваться какой-то стадией. ЕДИНЫЙ ЯЗЫК, который был разработан на ранних этапах проекта, должен развиваться на протяжении многих лет. Правда, этот процесс может в конечном счете потерять большое значение, но, вероятно, не скоро. Все это является частью стратегической инициативы вашей организации. Если не возложить на модель долгосрочные обязательства, то можно ли назвать то, над чем мы сегодня работаем, стратегическим термином СМЫСЛОВОЕ ЯДРО?

Архитектура

Есть еще один вопрос, который может вас заинтересовать. Что находится в ОГРАНИЧЕННОМ КОНТЕКСТЕ? Используя архитектурную диаграмму ПОРТЫ И АДАПТЕРЫ [IDDD], можно убедиться, что в ОГРАНИЧЕННЫЙ КОНТЕКСТ входит не только модель предметной области.

Следующие уровни являются типичными для ОГРАНИЧЕННОГО КОНТЕКСТА: АДАПТЕРЫ ВВОДА (например, контроллеры пользовательского интерфейса, конечные точки REST и слушатели сообщений); СЛУЖБЫ ПРИЛОЖЕНИЯ, оркестрирующие сценарии использования и управляющие транзакциями; модель предметной области, на которой мы сосредоточились; и АДАПТЕРЫ ВЫВОДА (например, механизмы хранения данных и рассылки сообщений). Об уровнях этой архитектуры можно долго рассуждать, но это неуместно в такой небольшой книге. Исчерпывающее обсуждение этой темы можно найти в главе 4 книги *Implementing Domain-Driven Design* [IDDD].

Модель предметной области, свободная от технологии

Несмотря на то что обсуждение архитектуры связано с технологией, модель предметной области должна быть свободной от технологии. С одной стороны, именно поэтому транзакции управляются службами приложения, а не моделью предметной области.

ПОРТЫ И АДАПТЕРЫ можно использовать как базовую архитектуру, но это не единственная архитектура, которую можно использовать вместе с подходом DDD. Наряду с архитектурой ПОРТЫ И АДАПТЕРЫ в рамках DDD можно применять любой из нижеперечисленных архитектурных стилей и архитектурных шаблонов (наряду с другими), смешивая и сравнивая их по мере необходимости.

- АРХИТЕКТУРА, УПРАВЛЯЕМАЯ СОБЫТИЯМИ; ИСТОЧНИКИ СОБЫТИЙ [IDDD]. Архитектура ИСТОЧНИКИ СОБЫТИЙ обсуждается в главе 6.

- РАЗДЕЛЕНИЕ ОТВЕТСТВЕННОСТИ НА ЗАПРОСЫ И КОМАНДЫ (CQRS — COMMAND QUERY RESPONSIBILITY SEGREGATION) [IDDD].
- РЕАКТИВНАЯ МОДЕЛЬ и МОДЕЛЬ АКТОРА; см. книгу Reactive Messaging Patterns with the Actor Model [Reactive], в которой описано использование МОДЕЛИ АКТОРА в рамках DDD.
- ПЕРЕДАЧА СОСТОЯНИЯ ПРЕДСТАВЛЕНИЯ (Representational State Transfer — REST) [IDDD].
- СЕРВИС-ОРИЕНТИРОВАННАЯ АРХИТЕКТУРА (Service-Oriented Architecture — SOA) [IDDD].
- МИКРОСЛУЖБЫ описываются в книге *Building Microservices* [Microservices]. Это эквивалент ОГРАНИЧЕННЫХ КОНТЕКСТОВ в подходе DDD, поэтому для их понимания достаточно прочитать эту книгу и *Implementing Domain-Driven Design* [IDDD].
- ОБЛАЧНЫЕ ВЫЧИСЛЕНИЯ поддерживаются так же, как микрослужбы, так что все, что вы прочтете в этой книге, а также в книгах *Implementing Domain-Driven Design* [IDDD] и *Reactive Messaging Patterns with the Actor Model* [Reactive], вполне применимо и к ним.

И еще один комментарий по поводу микрослужб. Некоторые специалисты считают, что микрослужба намного меньше, чем ОГРАНИЧЕННЫЙ КОНТЕКСТ. По их мнению, микрослужба моделирует только одну концепцию и управляет только одним узким типом данных. Примерами таких микрослужб являются классы `Product` и `BacklogItem`. Если вы считаете такую степень детализации достаточной для определения микрослужб, то учтите, что классы `Product` и `BacklogItem` находятся в одном и том же более крупном ОГРАНИЧЕННОМ КОНТЕКСТЕ. Два маленьких компонента микрослужбы отличаются друг от друга лишь разными модулями развертывания, которые могут влиять на их взаимодействие (см. КАРТЫ КОНТЕКСТОВ). Лингвистически они все еще находятся в пределах одних и тех же контекстных и семантических границ, основанных на методологии Scrum.

Резюме

В этой главе вы узнали:

- что одна из основных ошибок проектирования заключается в том, что разработчики включают слишком много компонентов в одну модель и создают БОЛЬШОЙ КОМ ГРЯЗИ;
- как использовать стратегическое проектирование DDD;
- как использовать ОГРАНИЧЕННЫЙ КОНТЕКСТ и ЕДИНЫЙ ЯЗЫК;

- как проверить свои предположения и объединить ментальные модели;
- как разработать ЕДИНЫЙ ЯЗЫК;
- какие архитектурные компоненты содержатся в ОГРАНИЧЕННОМ КОНТЕКСТЕ;
- что DDD не слишком труден для применения на практике!

Для более глубокого изучения ОГРАНИЧЕННЫХ КОНТЕКСТОВ обратитесь к главе 2 книги *Implementing Domain-Driven Design* [IDDD].

Глава 3

Стратегическое проектирование с помощью ПОДОБЛАСТЕЙ

Работая над проектом DDD, вы всегда будете сталкиваться с многочисленными ОГРАНИЧЕННЫМИ КОНТЕКСТАМИ. Один из ОГРАНИЧЕННЫХ КОНТЕКСТОВ содержит СМЫСЛОВОЕ ЯДРО, а в других ОГРАНИЧЕННЫХ КОНТЕКСТАХ заключены различные ПОДОБЛАСТИ. В предыдущей главе была показана важность разделения моделей, каждая из которых использует свой ЕДИНЫЙ ЯЗЫК, и формирования многочисленных ОГРАНИЧЕННЫХ КОНТЕКСТОВ. В предыдущей диаграмме есть шесть ОГРАНИЧЕННЫХ КОНТЕКСТОВ и шесть ПОДОБЛАСТЕЙ. Поскольку при разработке модели использовались методы стратегического предметно-ориентированного проектирования, группы разработчиков достигли ее оптимального состава: по одной ПОДОБЛАСТИ в каждом ОГРАНИЧЕННОМ КОНТЕКСТЕ и один ОГРАНИЧЕННЫЙ КОНТЕКСТ на каждую

ПОДОБЛАСТЬ. Иначе говоря, ЯДРО УПРАВЛЕНИЯ ГИБКИМ ПРОЕКТИРОВАНИЕМ представляет собой и один точно определенный ОГРАНИЧЕННЫЙ КОНТЕКСТ, и одну точно определенную ПОДОБЛАСТЬ. В редких ситуациях могут возникнуть многочисленные ПОДОБЛАСТИ в одном ОГРАНИЧЕННОМ КОНТЕКСТЕ, но это не позволяет получить оптимальный результат моделирования.

Что такое ПОДОБЛАСТЬ

Просто говоря, ПОДОБЛАСТЬ — это часть вашей предметной области. Вы можете думать о ПОДОБЛАСТИ как об отдельной логической модели некоторой предметной области. Большинство предметных областей обычно являются слишком большими и сложными, чтобы обсуждать их как единое целое, поэтому мы интересуемся только ПОДОБЛАСТЯМИ, которые должны использоваться в рамках отдельного проекта. ПОДОБЛАСТИ можно использовать для логического разделения предметной области, чтобы понять *пространство состояний* большого и сложного проекта.

Другой способ интерпретации ПОДОБЛАСТИ состоит в том, что она рассматривается как четко очерченная область компетенции, позволяющая найти решение для СМЫСЛОВОГО ЯДРА вашего бизнеса. Это означает, что в конкретной ПОДОБЛАСТИ есть один или несколько ЭКСПЕРТОВ ПРЕДМЕТНОЙ ОБЛАСТИ, которые очень хорошо понимают аспекты бизнеса, ведение которого облегчает данная ПОДОБЛАСТЬ. Кроме того, ПОДОБЛАСТЬ также имеет в некоторой степени стратегическое значение для вашего бизнеса.

Если бы для разработки ПОДОБЛАСТИ использовался подход DDD, то она была бы реализована как точно определенный ОГРАНИЧЕННЫЙ КОНТЕКСТ. ЭКСПЕРТАМИ ПРЕДМЕТНОЙ ОБЛАСТИ, которые специализируются в данной конкретной сфере бизнеса, были бы члены группы, разработавшей данный ОГРАНИЧЕННЫЙ КОНТЕКСТ. Использование DDD для разработки точно определенного ОГРАНИЧЕННОГО КОНТЕКСТА — оптимальный выбор, о котором иногда приходится только мечтать.

Типы ПОДОБЛАСТЕЙ

В рамках проекта есть три простых типа ПОДОБЛАСТЕЙ.

- СМЫСЛОВОЕ ЯДРО. Это область стратегических инвестиций в отдельную, точно определенную модель предметной области путем затрат существенных ресурсов для тщательной разработки ЕДИНОГО ЯЗЫКА в явном ОГРАНИЧЕННОМ КОНТЕКСТЕ. Эта задача имеет очень высокий приоритет для вашей организации, потому что именно ее решение дает вам конкурентное преимущество. Поскольку ваша организация

не может быть лидером во всем, что она делает, ваше СМЫСЛОВОЕ ЯДРО выделяет область ее конкурентного преимущества. Для того чтобы достичь такой глубины знаний, необходимы заинтересованность, сотрудничество и экспериментирование. Именно сюда организация должна вложить самые большие инвестиции, предназначенные для разработки программного обеспечения. Позднее мы рассмотрим средства ускорения и управления выполнением проектов.

- ВСПОМОГАТЕЛЬНАЯ ПОДОБЛАСТЬ. Такая ПОДОБЛАСТЬ требует специального моделирования ситуаций, для которых не существует готовых решений. Однако такие ПОДОБЛАСТИ не требуют настолько больших инвестиций, как СМЫСЛОВОЕ ЯДРО. Разработку таких ОГРАНИЧЕННЫХ КОНТЕКСТОВ можно поручить сторонним организациям, чтобы не перепутать их с областями стратегического назначения и не вложить в них слишком большие средства. И все же это важные программные модели, потому что ваше СМЫСЛОВОЕ ЯДРО не может успешно работать без них.
- УНИВЕРСАЛЬНАЯ ПОДОБЛАСТЬ. Этот вид решения можно приобрести готовым, но можно также привлечь к его разработке сторонние организации или даже отдельное подразделение вашей организации, в которую не входят элитные разработчики, которые в основном должны заниматься СМЫСЛОВЫМ ЯДРОМ и в меньшей степени ВСПОМОГАТЕЛЬНЫМИ ПОДОБЛАСТЯМИ. Опасайтесь принять УНИВЕРСАЛЬНУЮ ПОДОБЛАСТЬ за СМЫСЛОВОЕ ЯДРО. Не надо вкладывать в нее крупные инвестиции.

Обсуждая проект, разрабатываемый на основе принципов DDD, мы обычно обсуждаем его СМЫСЛОВОЕ ЯДРО.

Проблема сложности системы

Некоторые из системных границ в пределах предметной области с высокой вероятностью могут очерчивать унаследованные системы, которые ваша организация создала ранее или купила по лицензии. Вы не сможете намного улучшить унаследованные системы, но все же должны учитывать их, если они влияют на СМЫСЛОВОЕ ЯДРО проекта. Для этого в качестве инструмента для обсуждения вашего ПРОСТРАНСТВА СОСТОЯНИЙ необходимо использовать ПОДОБЛАСТИ.

К сожалению, некоторые унаследованные системы настолько неудобны для предметно-ориентированного проектирования на основе ОГРАНИЧЕННЫХ КОНТЕКСТОВ, что их можно было бы даже назвать *неограниченными* унаследованными системами. Такие унаследованные системы представляют собой то, что я уже назвал БОЛЬШИМ КОМОМ ГРЯЗИ. В действительности система может состоять из многочисленных запутанных моделей, которые следовало бы проектировать и реализовывать по отдельности, но они были смешаны вместе в одно очень сложное и переплетенное месиво.

Иначе говоря, когда мы обсуждаем унаследованную систему, то в ней, вероятно, существует несколько, а может, даже много, *логических* моделей предметной области. Каждую из этих логических моделей предметной области можно рассматривать как ПОДОБЛАСТЬ. На диаграмме каждая логическая ПОДОБЛАСТЬ в неограниченном унаследованном монолитном БОЛЬШОМ КОМЕ ГРЯЗИ обведена пунктиром. На рисунке показаны пять логических моделей, или ПОДОБЛАСТЕЙ. Обработка логических ПОДОБЛАСТЕЙ также помога-

ет справиться со сложностью больших систем. Это очень полезно, поскольку позволяет нам работать с *пространством задач* так, будто оно состоит из многочисленных ОГРАНИЧЕННЫХ КОНТЕКСТОВ, разработанных на основе принципов DDD.

Унаследованная система кажется менее монолитной и запутанной, если мы представим себе разные ЕДИНЫЕ ЯЗЫКИ, хотя бы ради понимания того, как нам их интегрировать в новую систему. Размышления и обсуждения такой унаследованной системы на основе ПОДОБЛАСТИ помогают нам справиться со сложностью большой запутанной модели. И поскольку мы рассуждаем, используя этот инструмент, мы можем выделить ПОДОБЛАСТИ, которые имеют большую ценность для бизнеса и необходимы для нашего проекта, и те, которые не заслуживают большого внимания.

Учитывая сказанное, вы можете даже показать СМЫСЛОВОЕ ЯДРО, над которым работаете или собираетесь работать, на той же самой простой диаграмме. Это поможет вам понять связи и зависимости между ПОДОБЛАСТЯМИ. Но я отложу детали этого обсуждения для КАРТ КОНТЕКСТОВ.

В рамках подхода DDD между ОГРАНИЧЕННЫМИ КОНТЕКСТАМИ и ПОДОБЛАСТЯМИ должно существовать взаимно однозначное соответствие. Таким образом, если в проекте DDD существует ОГРАНИЧЕННЫЙ КОНТЕКСТ, то в нем следует выделить одну модель ПОДОБЛАСТИ. Это может оказаться не всегда возможным или практичным, но к этому необходимо стремиться. Это сохранит ваши ОГРАНИЧЕННЫЕ КОНТЕКСТЫ точно определенными и позволит сосредоточиться на основной стратегической инициативе.

Если вы вынуждены создать вторую модель в том же самом ОГРАНИЧЕННОМ КОНТЕКСТЕ (в пределах вашего СМЫСЛОВОГО ЯДРА), то необходимо отделить вторичную модель от вашего ОСНОВНОГО ЯДРА, используя полностью отдельный МОДУЛЬ [IDDD]. (МОДУЛЬ DDD — это, как правило, пакет в языках Scala и Java или пространство имен в языках F# и C#.) Это

позволит ясно выразить, что одна модель — это ядро, а вторая носит вспомогательный характер. Это специфическое использование разделения на ПОДОБЛАСТИ вы можете использовать в своем *пространстве решений*.

Резюме

В этой главе вы узнали:

- что такое ПОДОБЛАСТИ и как они используются в ПРОСТРАНСТВЕ ПРОБЛЕМ и ПРОСТРАНСТВЕ РЕШЕНИЙ;
- чем различаются СМЫСЛОВОЕ ЯДРО, ВСПОМОГАТЕЛЬНАЯ ПОДОБЛАСТЬ и УНИВЕРСАЛЬНАЯ ПОДОБЛАСТЬ;
- как использовать ПОДОБЛАСТИ для интеграции с унаследованным БОЛЬШИМ КОМОМ ГРЯЗИ;
- что взаимно однозначное соответствие между ОГРАНИЧЕННЫМИ КОНТЕКСТАМИ DDD и их ПОДОБЛАСТЯМИ является очень важным;
- как отделить модель ВСПОМОГАТЕЛЬНОЙ ПОДОБЛАСТИ от модели СМЫСЛОВОГО ЯДРА с помощью МОДУЛЯ DDD в ситуациях, когда их непрактично разделять между двумя разными ОГРАНИЧЕННЫМИ КОНТЕКСТАМИ.

Исчерпывающее описание ПОДОБЛАСТЕЙ приведено в главе 2 книги *Implementing Domain-Driven Design* [IDDD].

Глава 4

Стратегическое проектирование на ОСНОВЕ СВЯЗЫВАНИЯ КОНТЕКСТОВ

В предыдущих главах было показано, что с каждым проектом DDD, помимо СМЫСЛОВОГО ЯДРА, связаны многочисленные ОГРАНИЧЕННЫЕ КОНТЕКСТЫ. Все понятия, которые не принадлежали Контексту управления гибким проектированием — СМЫСЛОВОМУ ЯДРУ, — относились к одному из остальных ОГРАНИЧЕННЫХ КОНТЕКСТОВ.

Кроме того, СМЫСЛОВОЕ ЯДРО управления гибким проектированием должно тесно взаимодействовать с другими ОГРАНИЧЕННЫМИ КОНТЕКСТАМИ. Эта интеграция в DDD называется СВЯЗЫВАНИЕМ КОНТЕКСТОВ. В предыдущей КАРТЕ КОНТЕКСТОВ вы могли увидеть, что концепция Обсуждение (`Discussion`) существует в обоих ОГРАНИЧЕННЫХ КОНТЕКСТАХ. Это объясняется тем, что источником класса `Discussion` является Контекст сотрудничества, а потребителем — Контекст управления гибким проектированием.

СВЯЗЫВАНИЕ КОНТЕКСТОВ отображено на диаграмме линией, окруженной пунктирным прямоугольником. (Этот прямоугольник не является частью контекста, а просто привлекает внимание к линии на рисунке.) Именно эта линия между двумя ограниченными контекстами создает КАРТУ КОНТЕКСТОВ. Иначе говоря, эта линия указывает на то, что два данных ОГРАНИЧЕННЫХ КОНТЕКСТА образуют некую карту. Следовательно, между этими двумя ОГРАНИЧЕННЫМИ КОНТЕКСТАМИ существует динамичное взаимодействие и определенная интеграция.

Учитывая, что в двух разных ОГРАНИЧЕННЫХ КОНТЕКСТАХ используются два ЕДИНЫХ ЯЗЫКА, эту линию можно интерпретировать как перевод между этими двумя языками. Представьте себе, что эти две группы должны сотрудничать, но они работают в разных странах и говорят на разных языках. Каждой из групп потребовался бы переводчик. В противном случае одна или обе группы должны были бы затратить много усилий на изучение языка другой группы. Проще найти переводчика, но это может оказаться слишком дорогим удовольствием. Например, представьте себе, сколько дополнительного времени требуется одной группе, чтобы переговорить с переводчиком, и сколько времени понадобится переводчику, чтобы передать сообщение другой группе. Это прекрасно работает в течение первых нескольких минут, но затем быстро утомляет. Впрочем, группы могли бы прийти к выводу, что лучше было бы выучить иностранный язык и постоянно переходить с одного языка на другой. Разумеется, мы описали отношения только между двумя группами. А что если в проекте участвуют несколько групп? Точно такие же компромиссы возникают при переводе одного ЕДИНОГО ЯЗЫКА на другой или при другом способе адаптации к иному ЕДИНОМУ ЯЗЫКУ.

Когда мы говорим о СВЯЗЫВАНИИ КОНТЕКСТОВ, нас интересует, *какие* отношения и *какая* интеграция между группами представлены линией, соединяющей любые два ОГРАНИЧЕННЫХ КОНТЕКСТА. Четкие границы и контракты между ними позволяют контролировать изменения в течение долгого времени. Есть несколько видов СВЯЗЫВАНИЯ КОНТЕКСТОВ, как групповые, так и технические, которые можно изобразить линиями. В некоторых случаях можно смешивать как отношения между группами, так и интеграцию.

Способы связывания контекстов

Какие отношения и какая интеграция могут быть представлены линией, связывающей контексты? Рассмотрим их подробнее.

ПАРТНЕРСТВО

Между двумя группами могут существовать отношения ПАРТНЕРСТВА. В этом случае каждая группа отвечает за один ОГРАНИЧЕННЫЙ КОНТЕКСТ. Группы создают ПАРТНЕРСТВО для согласования своих целей. Можно сказать, что обе группы либо вместе достигнут цели, либо вместе потерпят поражение. Так как они тесно связаны, их члены часто встречаются, чтобы синхронизировать календарные планы и взаимозависимую работу, а также поддерживать постоянную интеграцию для сохранения гармонии. Синхронизация представлена жирной сплошной линией, связывающей две группы. Эта линия указывает на очень высокий уровень взаимной зависимости.

Иногда ПАРТНЕРСТВО трудно поддерживать в течение долгого времени. поэтому многие группы, образовавшие ПАРТНЕРСТВО, устанавливают предельный срок действия отношений. ПАРТНЕРСТВО должно продолжаться только до тех пор, пока это приносит выгоду его участникам. После этого группы должны установить новые отношения.

ОБЩЕЕ ЯДРО

ОБЩЕЕ ЯДРО (SHARED KERNEL) на предыдущей диаграмме, образованное в результате пересечения двух ОГРАНИЧЕННЫХ КОНТЕКСТОВ, описывает отношения между двумя или более группами, которые совместно используют маленькую, но общую модель. Группы должны согласовать, какие элементы модели они собираются использовать совместно. Возможно, что только одна из групп обеспечивает создание, сборку и тестирование кода для общей модели. ОБЩЕЕ ЯДРО часто очень трудно выделить и обслуживать,

потому что для этого необходимо поддерживать открытое общение между группами и обеспечивать постоянное согласование общей модели. Однако такое сотрудничество может быть успешным, если все участвующие группы считают, что лучше использовать ОБЩЕЕ ЯДРО, чем ОТДЕЛЬНОЕ СУЩЕСТВОВАНИЕ (SEPARATE WAYS).

КЛИЕНТ-ПОСТАВЩИК

Отношение КЛИЕНТ-ПОСТАВЩИК (CLIENT-SUPPLIER) описывает взаимосвязи между двумя ОГРАНИЧЕННЫМИ КОНТЕКСТАМИ и соответствующими группами, где ПОСТАВЩИК — это вышестоящий контекст (обозначенный на диаграмме буквой U), а КЛИЕНТ — нижестоящий (обозначенный на диаграмме буквой D). ПОСТАВЩИК господствует в этих отношениях, потому что он должен обеспечить то, в чем нуждается КЛИЕНТ. КЛИЕНТ должен планировать сотрудничество с ПОСТАВЩИКОМ, чтобы достичь целей, но в конце концов именно ПОСТАВЩИК определяет, что и когда получит КЛИЕНТ. Это очень типичные и практичные отношения между группами, даже в пределах одной и той же организации, поскольку корпоративная культура не позволяет ПОСТАВЩИКУ быть полностью независимым и безразличным к реальным потребностям КЛИЕНТА.

КОНФОРМИСТ

Отношение КОНФОРМИСТ (CONFORMIST) возникает, когда есть вышестоящая и нижестоящая группы, и вышестоящая не имеет никакого стимула

для удовлетворения определенных потребностей нижестоящей группы. По различным причинам нижестоящая группа не может поддерживать усилия по переводу ЕДИНОГО ЯЗЫКА вышестоящей группы, чтобы удовлетворять свои потребности, и поэтому подстраивается под вышестоящую модель. Например, группа будет часто становиться КОНФОРМИСТОМ, объединяясь с очень большой, сложной и тщательно проработанной моделью. Например, группы, желающие интегрироваться с компанией Amazon в качестве ее аффилированных продавцов, часто подстраиваются под модель Amazon.com.

ПРЕДОХРАНИТЕЛЬНЫЙ УРОВЕНЬ

ПРЕДОХРАНИТЕЛЬНЫЙ УРОВЕНЬ (ANTICORRUPTION LAYER) — это отношение связывания контекстов, обеспечивающее их защиту. В этом случае нижестоящая группа создает уровень перевода между своим ЕДИНЫМ ЯЗЫКОМ (моделью) и ЕДИНЫМ ЯЗЫКОМ (моделью) вышестоящей группы. Этот уровень изолирует нижестоящую модель от вышестоящей и обеспечивает перевод сообщений между ними. Таким образом, это отношение также является разновидностью интеграции.

При любой возможности следует стремиться к созданию ПРЕДОХРАНИТЕЛЬНОГО УРОВНЯ между вашими нижестоящей и вышестоящей моделями, чтобы со своей стороны обеспечить возможность создания концепций модели, сохраняя их полную изолированность от внешних концепций. Впрочем, так же как и постоянное обеспечение перевода между двумя группами, говорящими на разных языках, стоимость такого уровня в некоторых случаях может оказаться слишком высокой.

СЛУЖБА С ОТКРЫТЫМ ПРОТОКОЛОМ

СЛУЖБА С ОТКРЫТЫМ ПРОТОКОЛОМ (OPEN HOST SERVICE) определяет протокол или интерфейс, предоставляющий доступ к вашему ОГРАНИЧЕННОМУ КОНТЕКСТУ с помощью ряда служб. Протокол является открытым, чтобы все, кто должен интегрироваться с вашим ОГРАНИЧЕННЫМ КОНТЕКСТОМ, могли использовать его относительно легко. Службы, предлагаемые интерфейсом прикладного программирования (API), хорошо задокументированы и удобны в использовании. Даже если вы относитесь к группе 2 на следующей диаграмме и у вас нет времени, чтобы обеспечить изоляцию своей стороны интеграции с помощью ПРЕДОХРАНИТЕЛЬНОГО УРОВНЯ, в этой модели вам много проще стать КОНФОРМИСТОМ, чем многочисленным унаследованным системам, с которыми вы можете столкнуться. Можно сказать, что язык СЛУЖБЫ С ОТКРЫТЫМ ПРОТОКОЛОМ много проще, чем язык систем другого типа.

ОБЩЕДОСТУПНЫЙ ЯЗЫК

ОБЩЕДОСТУПНЫЙ ЯЗЫК (PUBLISHED LANGUAGE), показанный на предыдущей диаграмме, — это хорошо документированный язык информационного обмена, допускающий простое употребление и перевод для любого количества ОГРАНИЧЕННЫХ КОНТЕКСТОВ. Потребители, читающие и пишущие на общедоступном языке, могут осуществлять прямой и обратный перевод с общего языка на общедоступный (и наоборот) с полной уверенностью, что их интеграция является правильной. Такой ОБЩЕДОСТУПНЫЙ ЯЗЫК может быть определен с помощью языков XML Schema, JSON Schema или более специальных языков наподобие Protobuf или Avro. Часто СЛУЖБЫ С ОТКРЫТЫМ ПРОТОКОЛОМ поддерживают и используют ОБЩЕДОСТУПНЫЙ ЯЗЫК, чтобы обеспечить лучшую интеграцию для третьих сторон. Эта комбинация делает очень удобным перевод между двумя ЕДИНЫМИ ЯЗЫКАМИ.

ОТДЕЛЬНОЕ СУЩЕСТВОВАНИЕ

ОТДЕЛЬНОЕ СУЩЕСТВОВАНИЕ (SEPARATE WAYS) описывает ситуацию, в которой интеграция с одним или несколькими ОГРАНИЧЕННЫМИ КОНТЕКСТАМИ не приносит большой выгоды за счет использования разных ЕДИНЫХ ЯЗЫКОВ. Возможно, функциональные возможности, которые вы ищете, не обеспечиваются полностью ни одним ЕДИНЫМ ЯЗЫКОМ. В этом случае вы разрабатываете свое собственное специализированное решение в своем ОГРАНИЧЕННОМ КОНТЕКСТЕ.

БОЛЬШОЙ КОМ ГРЯЗИ

Вы уже многое узнали о БОЛЬШОМ КОМЕ ГРЯЗИ в предыдущих главах, но я собираюсь вновь описать серьезные проблемы, с которыми вы столкнетесь, если будете вынуждены работать или интегрироваться с ним. Создания своего собственного БОЛЬШОГО КОМА ГРЯЗИ нужно избегать как чумы.

Если моего предупреждения вам недостаточно, то напомню, что случается в течение долгого времени, если вы все же создали БОЛЬШОЙ КОМ ГРЯЗИ: 1) возрастающее количество АГРЕГАТОВ загрязняет друг друга из-за незащищенных связей и зависимостей; 2) вмешательство в одну часть БОЛЬШОГО КОМА ГРЯЗИ вызывает рябь во всей модели, вызывая лавину проблем; 3) только коллективная мудрость и способность говорить сразу на всех языках может спасти систему от полного краха.

Проблема состоит в том, что в большом мире программных систем уже существует множество БОЛЬШИХ КОМОВ ГРЯЗИ, и их количество, без сомнения, будет расти каждый месяц. Даже если вы в состоянии избежать создания своего БОЛЬШОГО КОМА ГРЯЗИ, используя методики DDD, вы все же можете столкнуться с необходимостью интегрироваться с одним или несколькими БОЛЬШИМИ КОМАМИ ГРЯЗИ. В этом случае следует попытаться создать ПРЕДОХРАНИТЕЛЬНЫЙ СЛОЙ, изолирующий вас от каждой из унаследованных систем, чтобы защитить вашу модель от информационного мусора, кото-

рый иначе может превратить вашу модель в невообразимое болото. Что бы вы ни делали, не говорите на этом языке!

Правильное использование связывания контекстов

Вы можете задать вопрос: какой конкретный вид интерфейса позволяет осуществить интеграцию с заданным ОГРАНИЧЕННЫМ КОНТЕКСТОМ? Это зависит от группы, владеющей ОГРАНИЧЕННЫМ КОНТЕКСТОМ. Это может быть удаленный вызов процедур (Remote Procedure Call — RPC) по протоколу SOAP, или интерфейсы RESTful с ресурсами, или интерфейс рассылки сообщений, или механизм ИЗДАТЕЛЬ-ПОДПИСЧИК (PUBLISH-SUBSCRIBE). В крайнем случае можно использовать интеграцию базы данных или файловой системы, но будем надеяться, что этого никогда не случится. Интеграции базы данных действительно нужно избегать, но если вы все же вынуждены будете свернуть на этот путь, то сначала должны убедиться, что изолировали свою модель с помощью ПРЕДОХРАНИТЕЛЬНОГО УРОВНЯ.

Рассмотрим три наиболее надежных типа интеграции и будем двигаться от наименее надежного к самому надежному. Сначала рассмотрим подход RPC, затем RESTful HTTP и наконец рассылку сообщений.

Удаленный вызов процедур по протоколу SOAP

Удаленные вызовы процедур, или RPC, могут работать множеством способов. Один из наиболее популярных способов использования RPC — простой протокол доступа к объектам (Simple Object Access Protocol — SOAP)}. Идея, лежащая в основе использования механизма RPC с протоколом SOAP, заключается в том, чтобы сделать использование службы из другой системы похожей на вызов простой локальной процедуры или вызов метода. И все же запрос SOAP должен пройти по сети, достигнуть отдаленной системы, успешно выполниться и вернуть результаты по сети. В итоге возникает возможность полного отказа сети или, по крайней мере, непредвиденной задержки при первой попытке интеграции. Кроме того, механизм RPC с протоколом SOAP также подразумевает сильную связь между ОГРАНИЧЕННЫМ КОНТЕКСТОМ клиента и ОГРАНИЧЕННЫМ КОНТЕКСТОМ, обеспечивающим работу службы.

Основная проблема, связанная с использованием механизма RPC на основе протокола SOAP или другого протокола, заключается в том, что этот механизм может оказаться ненадежным. Если возникнет проблема с сетью или системой, являющейся главным интерфейсом прикладного программирования SOAP, то ваш, на первый взгляд, простой вызов процедуры потерпит крах, выдавая только ошибочные результаты. Не заблуждайтесь относительно мнимой легкости его использования!

Когда механизм RPC работает, — а в большинстве случаев он работает, — он является очень полезным способом интеграции. Если вы можете влиять на дизайн ОГРАНИЧЕННОГО КОНТЕКСТА службы, то было бы в ваших интересах иметь хорошо спроектированный интерфейс API, предоставляющий СЛУЖБУ С ОТКРЫТЫМ ПРОТОКОЛОМ и ОБЩЕДОСТУПНЫМ ЯЗЫКОМ. В любом случае ваш ОГРАНИЧЕННЫЙ КОНТЕКСТ клиента можно спроектировать с ПРЕДОХРАНИТЕЛЬНЫМ УРОВНЕМ, чтобы изолировать вашу модель от нежелательного внешнего влияния.

Протокол RESTful HTTP

Интеграция с помощью протокола RESTful HTTP сосредоточивает внимание на ресурсах, которыми обмениваются ОГРАНИЧЕННЫЕ КОНТЕКСТЫ, а также на четырех элементарных операциях: POST, GET, PUT и DELETE. Многие считают, что подход REST хорош для интеграции, потому что он

помогает определять хорошие интерфейсы API для распределенных вычислений. Трудно возражать против этого утверждения, учитывая успех Интернета и веб.

Существует вполне определенный образ мышления проектировщиков, которые используют протокол RESTful HTTP. Я не буду углубляться в детали в этой книге, но вы должны изучить этот протокол, прежде чем использовать подход REST. Для начала полезно прочитать книгу *REST in Practice* [RiP].

ОГРАНИЧЕННЫЙ КОНТЕКСТ службы, предоставляющий интерфейс REST, должен предоставить СЛУЖБУ С ОТКРЫТЫМ ПРОТОКОЛОМ и ОБЩЕДОСТУПНЫЙ ЯЗЫК. Ресурсы, определяемые в ОБЩЕДОСТУПНОМ ЯЗЫКЕ, в сочетании с идентификаторами REST URI формируют естественную СЛУЖБУ С ОТКРЫТЫМ ПРОТОКОЛОМ.

Протокол RESTful HTTP дает сбои практически по тем же причинам, по которым это делает механизм RPC, — отказы сети и провайдера или непредвиденно долгое время ожидания. Однако протокол RESTful HTTP основан на использовании Интернета, а кто может предъявить претензии к веб, если речь идет о надежности, масштабируемости и общем успехе проекта?

Типичная ошибка, которую делают, используя REST, состоит в проектировании ресурсов, которые непосредственно отражают АГРЕГАТЫ в модели предметной области. В результате все клиенты вступают в отношения КОНФОРМИСТ, в которых изменение модели приводит к изменениям ресурсов. Следовательно, этого делать нельзя. Вместо этого ресурсы необходимо проектировать искусственно, следуя за сценарием использования клиента. Слово "искусственно" означает, что клиенты получают ресурсы в том виде и составе, в котором они им нужны, а не в виде реальной модели предметной области. Иногда модель выглядит точно так, как хочет клиент. Но на дизайн ресурсов влияют желания клиента, а не текущий вид модели.

РАССЫЛКА СООБЩЕНИЙ

Используя для интеграции асинхронную передачу сообщений, ОГРАНИЧЕННЫЙ КОНТЕКСТ клиента может многого достичь, подписавшись на СОБЫТИЯ ПРЕДМЕТНОЙ ОБЛАСТИ, издаваемые вашим собственным или другим

ОГРАНИЧЕННЫМ КОНТЕКСТОМ. Рассылка сообщений — одна из наиболее устойчивых форм интеграции, потому что с ее помощью удаляется большая часть временных связей, возникающих из-за таких механизмов блокировки, как RPC и REST. Поскольку вы заранее ожидаете задержки сообщений, то можете создавать более устойчивые системы, так как никогда не ожидаете немедленных результатов.

Асинхронная передача данных с помощью механизма REST

Асинхронную передачу сообщений можно обеспечить с помощью опроса постепенно нарастающего набора ресурсов по протоколу REST. Используя фоновый процесс обработки, клиент может непрерывно запрашивать ресурсы у канала Atom, обеспечивающего постоянно увеличивающийся набор СОБЫТИЙ ПРЕДМЕТНОЙ ОБЛАСТИ. Это безопасный подход к поддержанию асинхронных операций между службой и клиентами, поскольку в службу непрерывно поступают новые события. Если служба по каким-то причинам становится недоступной, клиенты просто повторяют попытку через заданный интервал времени или выходят из системы и повторяют запросы, пока канал ресурсов подачи не станет доступным снова.

Этот подход подробно обсуждается в книге *Implementing Domain-Driven Design* [IDDD].

Не пускайте интеграцию под откос

Когда ОГРАНИЧЕННЫЙ КОНТЕКСТ клиента (C1) интегрируется с ОГРАНИЧЕННЫМ КОНТЕКСТОМ службы (S1), он не должен посылать синхронные блокирующие запросы к контексту S1 как прямой результат обработки запроса, посланного ему ранее. Иначе говоря, когда другой клиент (C0) посылает блокирующий запрос к контексту C1, не позволяйте контексту C1 блокировать запрос к контексту S1. Иначе возникает высокая вероятность того, чтобы интеграция между контекстами C0, C1 и S1 будет разрушена. Этого можно избежать с помощью асинхронной рассылки сообщений.

Как правило, АГРЕГАТ в одном ОГРАНИЧЕННОМ КОНТЕКСТЕ публикует СОБЫТИЕ ПРЕДМЕТНОЙ ОБЛАСТИ, которое могут получать многие заинтересованные стороны. Когда ОГРАНИЧЕННЫЙ КОНТЕКСТ подписки получает СОБЫТИЕ ПРЕДМЕТНОЙ ОБЛАСТИ, на основе его типа и значения выполняется некоторое действие. Обычно это приводит к созданию нового или изменению существующего АГРЕГАТА в ОГРАНИЧЕННОМ КОНТЕКСТЕ получателя.

Являются ли КОНФОРМИСТАМИ получатели СОБЫТИЯ ПРЕДМЕТНОЙ ОБЛАСТИ?

Вы можете задать вопрос: как СОБЫТИЯ ПРЕДМЕТНОЙ ОБЛАСТИ могут использоваться другим ОГРАНИЧЕННЫМ КОНТЕКСТОМ, не вынуждая ОГРАНИЧЕННЫЙ КОНТЕКСТ получателя вступать в отношение КОНФОРМИСТ. Как рекомендуется в главе 13 книги *Implementing Domain-Driven Design* [IDDD], потребители не должны использовать типы событий (например, классы), определенные их издателем. Вместо этого они должны зависеть только от схемы событий, т.е. их ОБЩЕДОСТУПНОГО ЯЗЫКА. Вообще говоря, это значит, что, если события изданы в формате JSON или более экономичном формате объекта, потребитель должен использовать события, выполняя их синтаксический разбор, чтобы выяснить их атрибуты.

Разумеется, сказанное означает, что ОГРАНИЧЕННЫЙ КОНТЕКСТ подписки может всегда извлекать пользу из незапрашиваемых событий в ОГРАНИЧЕННОМ КОНТЕКСТЕ издателя. Впрочем, иногда ОГРАНИЧЕННЫЙ КОНТЕКСТ клиента должен активно посылать КОМАНДНОЕ СООБЩЕНИЕ (COMMAND MESSAGE) ОГРАНИЧЕННОМУ КОНТЕКСТУ, чтобы вызвать некоторое действие. В таких случаях ОГРАНИЧЕННЫЙ КОНТЕКСТ клиента все еще будет получать все результаты как опубликованное СОБЫТИЕ ПРЕДМЕТНОЙ ОБЛАСТИ.

Во всех случаях использования рассылки сообщений для интеграции качество полного решения будет сильно зависеть от качества выбранного механизма рассылки сообщений. Механизм рассылки сообщений должен поддерживать ДОСТАВКУ НЕ РЕЖЕ ОДНОГО РАЗА [Reactive], чтобы гарантировать, что все сообщения в конечном счете будут получены. Это также означает, что ОГРАНИЧЕННЫЙ КОНТЕКСТ подписки должен быть реализован ИДЕМПОТЕНТНЫМ ПОЛУЧАТЕЛЕМ (IDEMPOTENT RECEIVER) [Reactive].

ДОСТАВКА НЕ РЕЖЕ ОДНОГО РАЗА (AT-LEAST-ONCE DELIVERY) [Reactive] — это шаблон рассылки сообщений, в котором механизм рассылки периодически повторно передает данное сообщение. Это происходит при потере сообщения, а также если получатели медленно реагируют или не в состоянии подтвердить получение сообщения. Благодаря такому дизайну механизма рассылки сообщений сообщение может быть доставлено больше одного раза, даже несмотря на то, что отправитель посылал его только один раз. Однако это не должно быть проблемой, если получатель спроектирован с учетом такой возможности.

Если сообщение может быть доставлено неоднократно, то получатель должен быть спроектирован так, чтобы правильно реагировать на эту ситуацию. Шаблон ИДЕМПОТЕНТНЫЙ ПОЛУЧАТЕЛЬ [Reactive] описывает, как получатель запроса выполняет операцию таким образом, что это приводит к тому же самому результату, даже если это действие выполнялось многократно. Таким образом, если одно и то же сообщение было получено много раз, то получатель отреагирует на это вполне безопасным способом. Это может означать, что получатель использует дедупликацию и проигнорирует повторное сообщение или безопасно повторно выполнит операцию с теми же самыми результатами, что и ранее.

Вследствие того что механизмы рассылки сообщений всегда вводят асинхронные связи ЗАПРОС-ОТВЕТ (REQUEST-RESPONSE) [Reactive], некоторая задержка является обычной и ожидаемой. Запросы к службе никогда (почти) не должны блокироваться во время работы службы. Таким образом,

проектирование механизма рассылки означает, что вы всегда будете планировать какую-то задержку, что сделает ваше решение более устойчивым с самого начала.

Пример связывания контекстов

Возвращаясь к примеру, рассмотренному в главе 2, зададим вопрос: где должен находиться тип `Policy`? Напомним, что в этом примере есть три разных типа `Policy` в трех разных ОГРАНИЧЕННЫХ КОНТЕКСТАХ. Итак, в какой ОГРАНИЧЕННЫЙ КОНТЕКСТ необходимо включить политику заключения договоров в страховой компании? Возможно, она относится к отделу заключения договоров, поскольку именно там она возникает. Для примера допустим, что она действительно относится к отделу заключения договоров. Тогда как другие ОГРАНИЧЕННЫЕ КОНТЕКСТЫ узнают о ее существовании?

Когда компонент типа Police возникает в Контексте заключения договоров, он может опубликовать СОБЫТИЕ ПРЕДМЕТНОЙ ОБЛАСТИ по имени PolicyIssued. С помощью подписки на сообщения любой другой ОГРАНИЧЕННЫЙ КОНТЕКСТ может отреагировать на это СОБЫТИЕ ПРЕДМЕТНОЙ ОБЛАСТИ. Эта реакция может включать в себя создание соответствующего компонента типа Policy в ОГРАНИЧЕННОМ КОНТЕКСТЕ подписки.

СОБЫТИЕ ПРЕДМЕТНОЙ ОБЛАСТИ PolicyIssued содержит идентификатор класса Police. В данном случае — это переменная policyId. Любые компоненты, созданные в ОГРАНИЧЕННОМ КОНТЕКСТЕ подписки, сохраняют этот идентификатор для отслеживания в Контексте заключения договоров. В этом примере идентификатор хранится в переменной issuedPolicyId. Если есть потребность в большем количестве типов Policy, чем может обеспечить СОБЫТИЕ ПРЕДМЕТНОЙ ОБЛАСТИ PolicyIssued, то ОГРАНИЧЕННЫЙ КОНТЕКСТ подписки может всегда сделать запрос к Контексту заключения договоров для получения дополнительной информации. В данном примере ОГРАНИЧЕННЫЙ КОНТЕКСТ подписки использует переменную issuedPolicyId, чтобы выполнить запрос к Контексту заключения договоров.

Компромисс между обогащением и обратными запросами

Иногда полезно обогатить СОБЫТИЯ ПРЕДМЕТНОЙ ОБЛАСТИ достаточно большим объемом данных, чтобы удовлетворить потребности всех потребителей. Иногда полезно хранить тонкие СОБЫТИЯ ПРЕДМЕТНОЙ ОБЛАСТИ и разрешать обратные запросы, когда потребители нуждаются в большем количестве данных. Первый вариант допускает большую автономию за-

висимых потребителей. Если автономия является вашим требованием, то рассмотрите возможность обогащения.

С другой стороны, трудно предсказать, какие части данных понадобятся потребителям СОБЫТИЙ ПРЕДМЕТНОЙ ОБЛАСТИ, и обогащение может оказаться крупным. Например, обогащение СОБЫТИЙ ПРЕДМЕТНОЙ ОБЛАСТИ может снизить безопасность. В таких случаях следует проектировать тонкие СОБЫТИЯ ПРЕДМЕТНОЙ ОБЛАСТИ и богатую модель запросов, гарантирующую безопасность потребителей.

Иногда обстоятельства вынуждают использовать смешанный подход.

А как выполняется обратный запрос к Контексту подписания договоров? Для этого в Контексте подписания договоров можно спроектировать СЛУЖБУ С ОТКРЫТЫМ ПРОТОКОЛОМ Restful и ОБЩЕДОСТУПНЫМ ЯЗЫКОМ. Найти переменную `IssuedPolicyData` можно с помощью простой команды HTTP `GET`.

Вас, вероятно, интересует, как устроены данные в СОБЫТИИ ПРЕДМЕТНОЙ ОБЛАСТИ `PolicyIssued`. Этот вопрос будет рассмотрен в главе 6.

Вас интересует, что случилось с примером Контекста управления гибким проектированием? Переключение на пример из области страхования позволил нам глубже исследовать подход DDD. Благодаря этому примеру вы должны еще лучше понимать принципы DDD. Не волнуйтесь, мы еще вернемся к Контексту управления гибким проектированием в следующей главе.

Резюме

В этой главе вы узнали:

- о различных видах СВЯЗЫВАНИЯ КОНТЕКСТОВ, таких как ПАРТНЕРСТВО, КЛИЕНТ-ПОСТАВЩИК и ПРЕДОХРАНИТЕЛЬНЫЙ УРОВЕНЬ;
- как использовать СВЯЗЫВАНИЕ КОНТЕКСТОВ для интеграции с механизмами RPC, RESTful HTTP и рассылки сообщений;
- как СОБЫТИЯ ПРЕДМЕТНОЙ ОБЛАСТИ работают с механизмом рассылки сообщений;
- на каком фундаменте основывается СВЯЗЫВАНИЕ КОНТЕКСТОВ.

Подробное описание КАРТ КОНТЕКСТОВ приведено в главе 3 книги *Implementing Domain-Driven Design* [IDDD].

Глава 5

Тактическое проектирование с помощью АГРЕГАТОВ

До сих пор мы рассматривали стратегическое проектирование с помощью ОГРАНИЧЕННЫХ КОНТЕКСТОВ, ПОДОБЛАСТЕЙ и КАРТ КОНТЕКСТОВ. На диаграмме вы видите два ОГРАНИЧЕННЫХ КОНТЕКСТА, СМЫСЛОВОЕ ЯДРО под названием Контекст управления гибким проектированием и Вспомогательную подобласть, предоставляющую инструменты взаимодействия с помощью СВЯЗЫВАНИЯ КОНТЕКСТОВ.

А что можно сказать о концепциях, которые находятся в ОГРАНИЧЕННОМ КОНТЕКСТЕ? Мы их уже упоминали, но настало время рассмотреть их более подробно. Они напоминают АГРЕГАТЫ в вашей модели.

Зачем нужны АГРЕГАТЫ

Каждая из концепций, окруженных эллипсами, которые вы видите в двух ОГРАНИЧЕННЫХ КОНТЕКСТАХ на диаграмме, является АГРЕГАТОМ. Одна концепция — Discussion — не окружена эллипсом. Она моделируется как ОБЪЕКТ-ЗНАЧЕНИЕ (OBJECT-VALUE). В этой главе мы сосредоточимся на АГРЕГАТАХ и подробно рассмотрим, как моделировать концепции Product, BacklogItem, Release и Sprint.

Что такое СУЩНОСТЬ

СУЩНОСТЬ (ENTITY) моделирует индивидуальный объект. Каждая СУЩНОСТЬ имеет уникальную идентичность, которая отличает ее от всех остальных СУЩНОСТЕЙ того же самого или другого типа. Очень часто, возможно, даже в подавляющем большинстве случаев, СУЩНОСТЬ является изменчивой, т.е. ее состояние с течением времени изменяется. Однако это не обязательно, и СУЩНОСТЬ может быть неизменной. Главная отличительная черта СУЩНОСТИ — ее уникальность, т.е. индивидуальность.

Исчерпывающее описание СУЩНОСТЕЙ приведено в книге *Implementing Domain-Driven Design* [IDDD].

Что такое АГРЕГАТ? На диаграмме представлены два типа АГРЕГАТОВ. Каждый АГРЕГАТ состоит из одной или нескольких СУЩНОСТЕЙ, одна из которых называется КОРНЕМ АГРЕГАТА. Кроме того, АГРЕГАТЫ могут содержать ОБЪЕКТЫ-ЗНАЧЕНИЯ. Как мы видим на диаграмме, ОБЪЕКТЫ-ЗНАЧЕНИЯ используются в обоих АГРЕГАТАХ.

Что такое ОБЪЕКТ-ЗНАЧЕНИЕ

ОБЪЕКТ-ЗНАЧЕНИЕ, или просто ЗНАЧЕНИЕ, моделирует неизменное концептуально цельное понятие. В пределах модели концепция ЗНАЧЕНИЕ представляет собой обычное значение. В отличие от СУЩНОСТИ, ЗНАЧЕНИЕ не имеет уникальной идентичности, а эквивалентность ЗНАЧЕНИЙ определяется с помощью сравнения атрибутов, инкапсулированных в типе ЗНАЧЕНИЯ. Кроме того, ОБЪЕКТ-ЗНАЧЕНИЕ — это не предмет, а средство определения количества или измерения СУЩНОСТИ.

Подробное описание объектов-значений см. в книге *Implementing Domain-Driven Design* [IDDD].

КОРНЕВАЯ СУЩНОСТЬ (ROOT ENTITY) каждого АГРЕГАТА владеет всеми остальными элементами, собранными в агрегате. Имя КОРНЕВОЙ СУЩНОСТИ — это концептуальное имя АГРЕГАТА. Это имя необходимо выбирать так, чтобы оно правильно описывало концепцию, которую моделирует АГРЕГАТ.

Каждый АГРЕГАТ формирует границу согласованности транзакций. Это значит, что, когда управляющая транзакция передается в базу данных, все составные части в пределах отдельного АГРЕГАТА должны быть непротиворечивыми и соответствовать бизнес-правилам. Это не означает, что после транзакции в АГРЕГАТЕ не должно быть элементов, которые могут оставаться несогласованными. В конце концов, АГРЕГАТ также моделирует концептуально цельное понятие. Но вы должны быть прежде всего заинтересованы в согласованности транзакций. Внешняя граница, нарисованная на диаграмме вокруг первого и второго типов АГРЕГАТА, представляет собой отдельную транзакцию, которая управляется механизмом атомарной персистентности каждого кластера.

Расширенная интерпретация транзакции

До некоторой степени использование транзакций в приложении — это деталь реализации. Например, как правило, в приложении существует ПРИКЛАДНАЯ СЛУЖБА [IDDD], управляющая атомарной транзакцией базы данных от имени модели предметной области. В другой архитектуре, например в модели АКТОР [Reactive], где каждый АГРЕГАТ реализуется в виде актора, транзакции можно обрабатывать, используя ИСТОЧНИКИ СОБЫТИЙ (см. следующую главу) и базу данных, которая не поддерживает атомарные транзакции. Как бы то ни было, под транзакцией я подразумеваю способ изоляции изменений АГРЕГАТА и поддержки бизнес-инвариантов — правил, которых всегда должно придерживаться программное обеспечение, — чтобы обеспечить согласованность следующей бизнес-операции. Независимо от того, как осуществляется контроль этого требования — с помощью атомарной транзакции базы данных или других средств, —

состояние АГРЕГАТА, т.е. его представление средствами ИСТОЧНИКОВ СОБЫТИЙ, должно быть безопасным, гарантировать правильные переходы и поддерживаться постоянно.

Границы транзакции определяются бизнесом, потому что именно он определяет, какое состояние кластера является правильным в любой заданный момент времени. Иначе говоря, если АГРЕГАТ не был сохранен в целости и сохранности, то выполненная бизнес-операция, согласно бизнес-правилам, должна считаться некорректной.

Для того чтобы взглянуть на это с другой точки зрения, рассмотрим следующую ситуацию. Хотя здесь представлены два АГРЕГАТА, только один из них должен быть зафиксирован в отдельной транзакции. Это общее правило проектирования АГРЕГАТОВ: в одной транзакции можно модифицировать и фиксировать только один экземпляр АГРЕГАТА. Именно поэтому вы видите только экземпляр первого типа АГРЕГАТА в пределах транзакции. Вскоре мы рассмотрим другие правила проектирования АГРЕГАТОВ.

Любой другой АГРЕГАТ будет изменяться и фиксироваться в другой транзакции. Именно поэтому говорят, что АГРЕГАТ окружен границей согласованности транзакций. Таким образом, АГРЕГАТЫ необходимо проектировать так, чтобы обеспечить согласованность транзакций и их успешное выполнение. Как показано на диаграмме, экземпляром второго типа управляет транзакция, отделенная от экземпляра первого типа АГРЕГАТА.

Поскольку экземпляры этих двух АГРЕГАТОВ спроектированы так, что они изменяются в отдельных транзакциях, возникает вопрос: как обеспечить изменение экземпляра второго типа АГРЕГАТА с учетом изменений, сделанных в экземпляре первого типа АГРЕГАТА, на которые должна реаги-

ровать наша модель предметной области? Это хороший вопрос, и ответ на него мы дадим немного позже.

Следует запомнить основной пункт этого раздела: именно бизнес-правила определяют, что должно быть цельным, полным и непротиворечивым в конце отдельной транзакции.

Эмпирические правила проектирования АГРЕГАТОВ

Рассмотрим четыре основных правила проектирования АГРЕГАТОВ.

1. Защищайте бизнес-инварианты в границах АГРЕГАТА.
2. Проектируйте маленькие АГРЕГАТЫ.
3. Ссылайтесь на другие АГРЕГАТЫ только по идентификаторам.
4. Обновляйте другие АГРЕГАТЫ, руководствуясь принципом итоговой согласованности.

Конечно, эти правила не являются догмами, установленными подходом DDD. Они просто являются выражением здравого смысла, помогая эффективно проектировать АГРЕГАТЫ. Далее мы подробнее рассмотрим каждое из этих правил, чтобы понять, почему они должны применяться везде, где только можно.

Правило 1. Защищайте бизнес-инварианты в границах АГРЕГАТА

Правило 1 означает, что именно бизнес должен в конечном счете определять состав АГРЕГАТА, основываясь на том, какие концепции должны оставаться согласованными после совершения транзакции. В предыдущем при-

мере класс Product проектируется так, что в конце транзакции все образованные экземпляры ProductBacklogItem должны быть совместимыми и согласованными с корнем класса Product. Кроме того, класс Sprint проектируется так, чтобы в конце транзакции все образованные экземпляры класса CommittedBacklogItem были совместимыми и согласованными с корнем класса Sprint.

Правило 1 становится более понятным в другом примере. Здесь фигурирует АГРЕГАТ BacklogItem. Существует бизнес-правило, которое гласит: если во всех экземплярах класса Task поле hoursRemaining равно нулю, то АГРЕГАТ BacklogItem должен иметь состояние DONE. Таким образом, в конце транзакции этот очень конкретный бизнес-инвариант должен быть выполнен. Этого требует бизнес-правило.

Правило 2. Проектируйте маленькие АГРЕГАТЫ

Это правило подчеркивает, что каждый АГРЕГАТ должен занимать небольшую область оперативной памяти и иметь небольшую область видимости транзакции. В предыдущей диаграмме представленный АГРЕГАТ не маленький. Здесь класс Product содержит потенциально очень большую коллекцию экземпляров класса BacklogItem, большую коллекцию экземпляров класса Release и большую коллекцию экземпляров класса Sprint. В течение долгого времени эти коллекции могут увеличиваться до больших размеров, включая тысячи экземпляров класса BacklogItem и, вероятно, сотни экземпляров классов Release и Sprint. Такое проектирование следует считать очень плохим.

Однако, если разделить АГРЕГАТ Product на четыре отдельных АГРЕГАТА, мы получим маленькие АГРЕГАТЫ Product, BacklogItem, Release и Sprint. Они быстрее загружаются, занимают меньше памяти и быстрее удаляются. Возможно, наиболее важно, что эти АГРЕГАТЫ приносят успех намного чаще, чем крупные АГРЕГАТЫ, такие как АГРЕГАТ Product в предыдущем примере.

Следование этому правилу приносит дополнительную выгоду в том, что с каждым АГРЕГАТОМ будет проще работать, потому что каждая связанная задача может управляться отдельным разработчиком. Это также означает, что АГРЕГАТ будет более простым для тестирования.

Проектируя АГРЕГАТЫ, следует помнить о ПРИНЦИПЕ ЕДИНСТВЕННОЙ ОБЯЗАННОСТИ (SINGLE RESPONSIBILITY PRINCIPLE – SRP). Если ваш АГРЕГАТ пытается сделать слишком многое, значит, он нарушает принцип SRP, и это

свидетельствует о его большом размере. Спросите себя, например, сосредоточен ли ваш класс Product только на реализации продукта Scrum, или он пытается выполнять и другие функции. Какова причина для изменения класса Product: улучшение продукта Scrum или управление невыполненными задачами, выпусками и спринтами? Вы должны изменять класс Product только для того, чтобы сделать его хорошим продуктом Scrum.

Правило 3. Ссылайтесь на другие АГРЕГАТЫ только по идентификаторам

Теперь, когда мы разделили большой кластер Product на четыре меньших АГРЕГАТА, как на них сослаться при необходимости? Здесь мы следуем правилу 3: ссылайтесь на другие АГРЕГАТЫ только по идентификаторам. В этом примере мы видим, что классы BacklogItem, Release и Sprint ссылаются на класс Product, сохраняя поле ProductId. Это помогает сохранять АГРЕГАТЫ маленькими и предотвращает изменение нескольких АГРЕГАТОВ в рамках одной транзакции.

Кроме того, это обеспечивает небольшой размер и эффективность АГРЕГАТОВ, снижая требования к памяти и ускоряя загрузку из постоянного хранилища, а также помогает установить правило, запрещающее изменять экземпляры другого АГРЕГАТА в рамках одной транзакции. Если у вас есть только уникальные идентификаторы других АГРЕГАТОВ, то у вас нет простого способа получить прямую ссылку на объект.

Другая выгода от использования ссылки по идентификатору состоит в том, что ваш АГРЕГАТ можно легко сохранить в практически любом меха-

низме хранения данных, например, в реляционной или документной базе данных, хранилище типа "ключ–значение" и сетках/фабриках данных. Это означает, что вы можете использовать реляционные таблицы MySQL для хранения JSON-документов, например PostgreSQL или MongoDB, GemFire/Geode, Coherence и GigaSpaces.

Правило 4. Обновляйте другие АГРЕГАТЫ, руководствуясь принципом итоговой согласованности

В нашем примере экземпляр класса `BacklogItem` назначается для спринта, который представлен экземпляром класса `Sprint`. Экземпляры классов `BacklogItem` и `Sprint` должны реагировать на это событие. Экземпляр класса `BacklogItem` первым узнает, что он назначен для спринта. Это происходит в рамках одной транзакции, в которой изменяется состояние экземпляра класса `BacklogItem`, получающего идентификатор `SprintId` целевого экземпляра класса `Sprint`. Как же гарантировать, что этот экземпляр класса `Sprint` также получит значение идентификатора `BacklogItemId` предназначенного для него экземпляра класса `BacklogItem`?

В рамках транзакции АГРЕГАТ BacklogItem публикует СОБЫТИЕ ПРЕДМЕТНОЙ ОБЛАСТИ по имени BacklogItemCommitted. Когда транзакция BacklogItem завершается, ее состояние сохраняется вместе с СОБЫТИЕМ ПРЕДМЕТНОЙ ОБЛАСТИ ItemCommitted. Когда экземпляр класса BacklogItem Committed поступает к локальному подписчику, начинается транзакция и состояние экземпляра класса Sprint изменяется, чтобы сохранить идентификатор BacklogItemId назначенного экземпляра BacklogItem. Экземпляр класса Sprint сохраняет экземпляр класса BacklogItemId в новой СУЩНОСТИ CommittedBacklogItem.

Как было указано в главе 4, СОБЫТИЯ ПРЕДМЕТНОЙ ОБЛАСТИ издаются АГРЕГАТОМ. Заинтересованный ОГРАНИЧЕННЫЙ КОНТЕКСТ подписывается на их получение. Механизм передачи сообщений поставляет СОБЫТИЯ ПРЕДМЕТНОЙ ОБЛАСТИ заинтересованным сторонам с помощью инструментов подписки. Заинтересованный ОГРАНИЧЕННЫЙ КОНТЕКСТ может быть как тем же самым ОГРАНИЧЕННЫМ КОНТЕКСТОМ, в котором публикуется СОБЫТИЕ ПРЕДМЕТНОЙ ОБЛАСТИ, так и другим.

Например, в случае АГРЕГАТОВ BacklogItem и Sprint издатель и подписчик находятся в одном и том же ОГРАНИЧЕННОМ КОНТЕКСТЕ. В данном случае совершенно необязательно использовать разнообразные средства связующего программного обеспечения для рассылки сообщений, но это довольно легко сделать, поскольку эти инструменты используются для публикации событий, предназначенных для других ОГРАНИЧЕННЫХ КОНТЕКСТОВ.

Если итоговая согласованность кажется труднодостижимой

В использовании принципа итоговой согласованности (eventual consistency) нет ничего невероятно сложного. И все же, пока вы не наберетесь опыта, вы можете испытывать трудности. В этом случае необходимо разделить вашу модель на АГРЕГАТЫ согласно границам, определенным бизнес-правилами. Впрочем, ничто не мешает вам фиксировать модификации двух или более АГРЕГАТОВ в отдельной атомарной транзакции базы данных. Этот подход можно использовать в тех случаях, когда вы уверены в его успехе, но в остальных ситуациях следует полагаться на принцип итоговой согласованности. Это позволит вам привыкнуть к новым методикам, не тратя много времени на раскачку. Только следует иметь в виду, что это не основной способ, которым должны использоваться АГРЕГАТЫ, и в результате вы можете столкнуться с отказами транзакций.

Моделирование агрегатов

В работе над моделью предметной области вас поджидает несколько ловушек, связанных с реализацией АГРЕГАТОВ. Самая крупная и опасная ловушка — АНЕМИЧНАЯ МОДЕЛЬ ПРЕДМЕТНОЙ ОБЛАСТИ [IDDD]. Это объект-

но-ориентированная модель предметной области, в которой все АГРЕГАТЫ имеют только открытые методы доступа (get и set) и не содержат никаких реальных бизнес-функций. Это часто случается, когда моделирование носит более теоретический, чем практический характер. Проектирование АНЕМИЧНОЙ МОДЕЛИ ПРЕДМЕТНОЙ ОБЛАСТИ означает формальное применение модели предметной области без реализации ни одного из ее преимуществ. Не попадайтесь на эту приманку!

Также нельзя допускать утечку бизнес-логики из вашей модели предметной области в ПРИКЛАДНЫЕ СЛУЖБЫ. Это может свидетельствовать о скрытой анемии. Делегирование бизнес-логики от служб во вспомогательные/служебные классы тоже ни к чему хорошему не приведет. Служебные утилиты всегда демонстрируют кризис идентичности и никогда не могут работать правильно. Поместите вашу бизнес-логику в свою модель предметной области, или вам придется страдать от ошибок, вызванных АНЕМИЧНОЙ МОДЕЛЬЮ ПРЕДМЕТНОЙ ОБЛАСТИ.

Как насчет функционального программирования?

При использовании функционального программирования правила существенно изменяются. В то время как в объектно-ориентированном программировании АНЕМИЧНАЯ МОДЕЛЬ ПРЕДМЕТНОЙ ОБЛАСТИ — плохая идея, в области функционального программирования она является нормой. Это объясняется тем, что функциональное программирование стимулирует разделение данных и поведения. Ваши данные проектируются как неизменяемые структуры данных или типы записи, а поведение реализуется как функции без побочных эффектов, работающие с неизменяемыми записями определенных типов. Вместо того чтобы изменять данные, которые функции получают как параметры, функции возвращают новые значения. Эти новые значения могут быть новым состоянием АГРЕГАТА или СОБЫТИЕМ ПРЕДМЕТНОЙ ОБЛАСТИ, представляющим переход в состояние АГРЕГАТА.

В этой главе мы достаточно подробно остановились на объектно-ориентированном подходе, потому что он все еще наиболее широко используется и глубоко проработан. И все же, если вы используете функциональный язык и подход DDD, то должны знать, что часть материала этой книги к ним не применима или, по крайней мере, некоторые правила должны быть изменены.

Далее я собираюсь показать вам несколько технических компонентов, необходимых для разработки основных АГРЕГАТОВ. Я предполагаю, что вы используете Scala, C#, Java или другой объектно-ориентированный язык программирования. Следующие примеры написаны на C#, но они совершенно понятны всем программистам, работающим на языках Scala, F#, Java, Rubin и Python.

```
public class Product : Entity
{
  ...
}
```

Для начала необходимо создать класс для вашей КОРНЕВОЙ СУЩНОСТИ АГРЕГАТА. Ниже представлена диаграмма UML (Unified Modeling Language), описывающая КОРНЕВУЮ СУЩНОСТЬ `Product`. На ней также показан класс `Product` на языке C#, который расширяет базовый класс `Entity`. Этот базовый класс включает в себя несколько стандартных характеристик СУЩНОСТИ. Исчерпывающее описание вопросов, связанных с проектированием и реализацией СУЩНОСТЕЙ и АГРЕГАТОВ, см. в [IDDD].

```
public class Product : Entity
{
  private ProductId productId;
  private TenantId tenantId;
}
```

Каждая КОРНЕВАЯ СУЩНОСТЬ АГРЕГАТА должна иметь глобально уникальный идентификатор. Класс `Product` в Контексте управления гибким проек-

тированием фактически имеет два глобально уникальных идентификатора. С помощью идентификатора TenantId область видимости КОРНЕВОЙ СУЩНОСТИ ограничивается организацией подписчика. (Каждая организация, которая подписалась на предлагаемую службу, известна как арендатор и, таким образом, имеет уникальную идентичность.) Второй идентификатор, который также является глобально уникальным, — ProductId. Он отделяет экземпляр класса Product от всех других экземпляров в пределах одного и того же арендатора. Кроме того, на диаграмме показан код C#, объявляющий два идентификатора в классе Product.

Использование ОБЪЕКТОВ-ЗНАЧЕНИЙ

Здесь идентификаторы TenantId и ProductId смоделированы как неизменяемые Объекты-Значения.

```
public class Product : Entity
{
  private string description;
  private string name;
  private ProductId productId;
  private TenantId tenantId;
}
```

Затем необходимо описать все внутренние атрибуты, или поля, которые являются необходимыми для поиска АГРЕГАТА. У класса Product есть атрибуты description (описание) и name (имя). С помощью обоих этих атрибутов или одного из них пользователи могут найти любой экземпляр класса Product. Я также привожу код на C#, который объявляет эти два внутренних атрибута.

```
public class Product : Entity
{
  ...
  public string Description
    { get; private set; }

  public string Name
    { get; private set; }
}
```

Конечно, вы можете добавить простую функцию, например метод для чтения внутренних атрибутов (`get`). В языке C# это можно было бы сделать с помощью открытых свойств `get`. Однако методы `set` нельзя делать открытыми. А как использовать значения свойства или атрибута без открытых методов `set`? При использовании объектно-ориентированного подхода (C#, Scala, и Java) внутреннее состояние изменяют с помощью поведенческих методов. В рамках функционального подхода (F#, Scala и Clojure) функции возвращают новые значения, которые отличаются от значений, которые передаются как параметры.

```
public class Product : Entity
{
  ...
  public string Name
    { get; private set; }
}
```

Вы должны бороться с АНЕМИЧНОЙ МОДЕЛЬЮ ПРЕДМЕТНОЙ ОБЛАСТИ [IDDD]. Предоставляя открытые методы `set`, вы быстро получите анемичную модель, потому что в этом случае логика, в соответствии с которой устанавливаются значения атрибутов экземпляров класса `Product`, реализовалась бы вне модели. Хорошенько подумайте, прежде чем так поступать, и не забывайте об этом предупреждении!

```
public class Product : Entity
{
   ...
   public void PlannedProductBacklogItem(...)
   {
      ...
   }
}
```

Наконец, вы добавляете любые сложные функции. Здесь мы имеем четыре новых метода: `PlanBacklogItem()`, `PlannedProductBacklogItem()`, `ScheduleRelease()` и `ScheduleSprint()`. Код на языке C# для каждого из этих методов должен быть добавлен в класс.

Помните, что, используя DDD, мы всегда моделируем ЕДИНЫЙ ЯЗЫК в ОГРАНИЧЕННОМ КОНТЕКСТЕ. Таким образом, все части АГРЕГАТА `Product` моделируются на ЕДИНОМ ЯЗЫКЕ. Вы не просто создаете компоненты, вы организовываете гармоничное взаимодействие ЭКСПЕРТОВ ПРЕДМЕТНОЙ ОБЛАСТИ и разработчиков вашей сплоченной группы.

Тщательно выбирайте абстракции

Эффективная программная модель всегда основана на ряде абстракций, описывающих ведение бизнеса. Следовательно, необходимо правильно

выбирать соответствующий уровень абстракции для каждой моделируемой концепции.

Если вы будете подчиняться правилам ЕДИНОГО ЯЗЫКА, то будете создавать правильные абстракции. Намного проще правильно моделировать абстракции, если существуют ЭКСПЕРТЫ ПРЕДМЕТНОЙ ОБЛАСТИ, определяющие хотя бы основы вашего языка моделирования. Однако иногда не в меру старательные разработчики программного обеспечения пытаются решать неправильные проблемы с помощью абстракций, которые являются уж слишком абстрактными.

Например, в Контексте управления гибким проектированием мы имеем дело с методологией Scrum. В этом случае целесообразно моделировать концепции `Product`, `BacklogItem`, `Release` и `Sprint`, которые мы уже обсудили. А что делать, если разработчики программного обеспечения меньше интересуются моделированием ЕДИНОГО ЯЗЫКА Scrum и вместо этого стремятся моделировать все текущие и будущие концепции Scrum?

В таком случае разработчики, вероятно, придумали бы абстракции вроде `ScrumElement` и `ScrumElementContainer`. Концепция `ScrumElement` могла бы удовлетворить текущую потребность в концепции `Product` и `BacklogItem`, а концепция `ScrumElementContainer` могла бы представить гораздо более явные концепции `Release` и `Sprint`. Класс `ScrumElement` имел бы свойство `typeName`, которое в соответствующих ситуациях было бы установлено равным "`Product`" или "`BacklogItem`". Мы могли бы проектировать свойство `typeName` в классе `ScrumElementContainer` и позволить устанавливать его значения "`Release`" или "`Спринт`".

Вы видите проблемы, связанные с этим подходом? Их довольно много, но мы назовем лишь несколько.

- Язык программной модели не соответствует ментальной модели ЭКСПЕРТОВ ПРЕДМЕТНОЙ ОБЛАСТИ.
- Уровень абстракции слишком высокий, и вы получите большие неприятности, когда начнете моделировать детали каждого конкретного типа.
- Этот путь ведет к созданию специальных разновидностей в каждом из классов и, вероятно, к сложной иерархии классов с соответствующими проблемами.
- Вы получите намного больше кода, чем требуется, потому что пытаетесь решить неразрешимую проблему, которая не должна рассматриваться вообще.
- Часто язык неправильных абстракций отражается даже на пользовательском интерфейсе, который вызывает замешательство у пользователей.

- Вы впустую потратите значительные объемы времени и денег.
- Вы никогда не сможете предвидеть будущие потребности, т.е. если когда-либо в будущем в методологию Scrum будут включены новые концепции, ваша существующая модель не сможет их учесть.

Некоторым читателям может показаться странным такой образ действий, но выбор неправильного уровня абстракций довольно часто встречается в реализациях, создаваемых на основе чисто технических соображений.

Не поддавайтесь соблазну высоких абстракций. Моделируйте ЕДИНЫЙ ЯЗЫК в соответствии с ментальной моделью ЭКСПЕРТОВ ПРЕДМЕТНОЙ ОБЛАСТИ, которая уточняется вашей группой. Моделируя то, в чем бизнес нуждается сегодня, вы сэкономите значительный объем времени, бюджета и кода и создадите меньше проблем. Более того, вы окажете бизнесу большую услугу, моделируя точный и полезный ОГРАНИЧЕННЫЙ КОНТЕКСТ, отражающий эффективный дизайн.

Агрегаты правильного размера

Вы можете задать вопрос: как определить границы АГРЕГАТОВ и предотвратить создание больших кластеров, сохранив границы согласованности, защищающие истинные бизнес-инварианты? Ниже будут описаны методы проектирования, заслуживающие высокой оценки. Если вы уже создали крупные АГРЕГАТЫ, то можете использовать этот подход повторно для рефакторинга и создания АГРЕГАТОВ меньшего размера, но я не собираюсь рассматривать эти вопросы.

Рассмотрим следующие этапы проектирования, которые помогут вам обеспечить правильные границы согласованности.

1. Для начала вспомните второе правило проектирования агрегатов: проектируйте маленькие АГРЕГАТЫ. Начните с создания АГРЕГАТОВ, содержащих только одну СУЩНОСТЬ, которая будет служить КОРНЕМ АГРЕГАТА. Даже не думайте помещать две СУЩНОСТИ внутри отдельной границы. Эта возможность может появиться достаточно скоро. Заполните каждую из СУЩНОСТЕЙ полями/атрибутами/свойствами, которые вы считаете наиболее тесно связанными с единственной КОРНЕВОЙ СУЩНОСТЬЮ. Подсказка: здесь следует определить каждое поле/атрибут/свойство, которые требуются для идентификации и поиска АГРЕГАТА, а также любые дополнительные внутренние поля/атрибуты/свойства, необходимые для создания АГРЕГАТА, находящегося в правильном начальном состоянии.
2. Теперь вспомните первое правило проектирования агрегатов: защищайте бизнес-инварианты в границах АГРЕГАТА. На предыдущем

этапе вы уже установили значения всех внутренних полей/атрибутов, которые должны обновляться при сохранении единственной СУЩНОСТИ АГРЕГАТА. Но теперь вы должны рассмотреть каждый из АГРЕГАТОВ по отдельности. Например, рассматривая АГРЕГАТ A1, спросите ЭКСПЕРТОВ ПРЕДМЕТНОЙ ОБЛАСТИ, не должны ли все другие АГРЕГАТЫ, определенные вами, обновляться в ответ на изменения АГРЕГАТА A1. Создайте список АГРЕГАТОВ и правила их согласованности, которые будут указывать временные интервалы для всех обновлений в качестве реакции на изменения АГРЕГАТА A1. Иначе говоря, во главе этого списка стоял бы пункт "АГРЕГАТ A1", а другие типы АГРЕГАТОВ перечислялись бы ниже, если они будут обновляться в ответ на обновления АГРЕГАТА A1.

3. Спросите ЭКСПЕРТОВ ПРЕДМЕТНОЙ ОБЛАСТИ, сколько времени может пройти до каждого из обновлений в ответ на изменения АГРЕГАТА. Это приведет вас к двум видам спецификаций: а) немедленно, б) в течение *N* секунд/минут/часов/дней. Один из возможных способов найти правильный порог, ориентируясь на бизнес-правила, — установить чрезмерно большой интервал времени (например, недели или месяцы), что, очевидно, является недопустимым. Это, вероятно, вызовет возражения бизнес-экспертов и заставит их уточнить приемлемый интервал времени.
4. Для каждого из вариантов немедленной реакции (пункт 3*а*) вы должны тщательно рассмотреть возможность создания двух СУЩНОСТЕЙ внутри границы одного и того же АГРЕГАТА. Это означает, например, что АГРЕГАТ A1 и АГРЕГАТ A2 будут фактически объединены в новый АГРЕГАТ [1,2]. Теперь АГРЕГАТОВ A1 и A2 больше не будет. Останется только АГРЕГАТ [1,2].
5. Для каждого из реагирующих АГРЕГАТОВ, которые могут быть обновлены в течение заданного времени (пункт 3*б*), примените четвертое правило проектирования АГРЕГАТОВ: обновляйте другие АГРЕГАТЫ, руководствуясь принципом итоговой согласованности.

На этом рисунке в центре внимания находится моделирование АГРЕГАТА A1. Обратите внимание на список правил согласованности АГРЕГАТА A1. В нем указано, что АГРЕГАТ A2 реагирует немедленно, в то время как АГРЕГАТ C14 имеет для этого 30 секунд. В результате АГРЕГАТЫ A1 и A2 моделируются как единый АГРЕГАТ [1,2]. Во время выполнения программы АГРЕГАТ [1,2] публикует СОБЫТИЕ ПРЕДМЕТНОЙ ОБЛАСТИ, которое в конечном счете вызывает обновление АГРЕГАТА C14.

Будьте внимательны: бизнес-правила не требуют, чтобы каждый АГРЕГАТ подчинялся спецификации 3*a* (немедленная согласованность). Это может оказаться особенно сильной тенденцией, если проект зависит от дизайна базы данных и модели данных. Стороны, разработавшие базу данных и модели данных, будут очень сосредоточены на транзакциях. Однако крайне маловероятно, что в каждом случае бизнесу действительно необходима немедленная согласованность. Для того чтобы изменить этот образ мышления, вам, вероятно, придется потратить время, доказывая, что транзакции будут терпеть неудачу из-за параллельных обновлений многочисленными пользователями разных частей АГРЕГАТА, который станет крупным кластером. Кроме того, вы можете указать, сколько дополнительной памяти потребуется для создания большого кластера. Очевидно, этих проблем следует избегать в первую очередь.

Это упражнение показывает, что итоговая согласованность зависит от вида бизнеса, а не от технических факторов. Конечно, вы должны будете найти технический способ обеспечить итоговую согласованность обновлений многочисленных АГРЕГАТОВ, как было сказано в предыдущей главе, посвященной СВЯЗЫВАНИЮ КОНТЕКСТОВ. Но даже в этом случае только бизнес может определить приемлемый интервал времени для обновлений разных СУЩНОСТЕЙ. Некоторые из них должны обновляться немедленно, в рамках транзакции, т.е. они должны управляться тем же самым АГРЕГАТОМ. Другие могут обновляться в конечном счете, т.е. они могут управляться с помощью, например, СОБЫТИЙ ПРЕДМЕТНОЙ ОБЛАСТИ и рассылки сообщений. Размышления над тем, как бизнес мог выполнять эти операции на основе системы бумажного документооборота, может навести вас на ценные идеи о том, как разные операции, управляемые предметной областью, должны работать в пределах программной модели бизнес-операций.

Тестируемые модули

Вы должны проектировать ваши АГРЕГАТЫ так, чтобы обеспечить надежную инкапсуляцию для модульного тестирования. Сложные АГРЕГАТЫ трудно проверять. Следуя правилам проектирования АГРЕГАТОВ, вы сможете моделировать тестируемые АГРЕГАТЫ.

Как показано в главах 2 и 7, модульное тестирование отличается от проверки бизнес-спецификаций (приемочных тестов). Разработка модульных тестов приводит к созданию сценариев приемочных тестов на основе спецификаций. В ходе этих проверок мы должны убедиться в том, что АГРЕГАТ работает именно так, как запланировано. Для гарантии правильности, качества и надежности ваших АГРЕГАТОВ необходимо выполнить все операции. Для этого можно использовать модульное тестирование, которому посвящено много книг и статей. Модульные тесты будут непосредственно связаны с вашим ОГРАНИЧЕННЫМ КОНТЕКСТОМ и храниться вместе с архивом исходных кодов.

Резюме

В этой главе вы узнали:

- что такое шаблон АГРЕГАТ и почему вы должны использовать его;
- о важности проектирования с учетом границ согласованности;
- из чего состоит АГРЕГАТ;
- четыре эмпирических правила эффективного проектирования АГРЕГАТОВ;
- как моделировать уникальную идентичность АГРЕГАТА;
- важность атрибутов АГРЕГАТА и как предотвратить создание АНЕМИЧНОЙ МОДЕЛИ ПРЕДМЕТНОЙ ОБЛАСТИ;
- как моделировать поведение АГРЕГАТА, чтобы всегда придерживаться ЕДИНОГО ЯЗЫКА в пределах ОГРАНИЧЕННОГО КОНТЕКСТА;
- что для вашего проекта важен выбор надлежащего уровня абстракции;
- как правильно устанавливать размеры АГРЕГАТОВ и обеспечить их тестируемость.

Для более всестороннего изучения СУЩНОСТЕЙ, ОБЪЕКТОВ-ЗНАЧЕНИЙ и АГРЕГАТОВ см. главы 5, 6 и 10 книги *Implementing Domain-Driven Design* [IDDD].

Глава 6

Тактическое проектирование с помощью СОБЫТИЙ ПРЕДМЕТНОЙ ОБЛАСТИ

В предыдущих главах мы уже несколько раз показывали, как используются СОБЫТИЯ ПРЕДМЕТНОЙ ОБЛАСТИ, представляющие собой запись о некоторой ситуации, возникшей в ОГРАНИЧЕННОМ КОНТЕКСТЕ и имеющей определенное значение для бизнеса. К настоящему времени вы уже знаете, что СОБЫТИЕ ПРЕДМЕТНОЙ ОБЛАСТИ — очень важный инструмент стратегического проектирования. Однако часто в ходе тактического проектирования СОБЫТИЯ ПРЕДМЕТНОЙ ОБЛАСТИ схематизируются и становятся частью вашего СМЫСЛОВОГО ЯДРА.

Для того чтобы увидеть мощь СОБЫТИЙ ПРЕДМЕТНОЙ ОБЛАСТИ, рассмотрим концепцию причинной согласованности. Предметная область обеспечивает причинную согласованность, если порядок ее причинно связанных опе-

раций, т.е. операций, одна из которых вызывает другую, сохраняется в каждом зависимом узле распределенной системы [Causal]. Это означает, что причинно связанные операции должны выполняться в определенном порядке, и, следовательно, событие не может произойти, пока не произойдет предшествующее ему событие. В частности, это может означать, что один АГРЕГАТ не может быть создан или изменен, пока к другому АГРЕГАТУ не будет применена определенная операция.

1. Сью публикует сообщение: "Я потеряла бумажник!"
2. Гэри говорит в ответ: "Это ужасно!"
3. Сью публикует сообщение: "Не волнуйся, я нашла бумажник!"
4. Гэри отвечает: "Очень хорошо!"

Если бы эти сообщения были скопированы в распределенные узлы без учета причинной согласованности, то могло бы оказаться, что Гэри сказал: "Очень хорошо!" в ответ на сообщение "Я потеряла бумажник!" Сообщение "Очень хорошо!" не имеет причинно-следственной связи с выражением "Я потеряла бумажник!", и совершенно ясно, что это не то сообщение, которое Гэри хотел опубликовать в ответ на сообщение Сью. Следовательно, если причинно-следственная связь не обеспечена надлежащим способом, вся предметная область может оказаться неправильной или, по крайней мере, запутанной. Этот вид причинной, линеаризованной системной архитектуры можно организовать с помощью создания и публикации правильно упорядоченных СОБЫТИЙ ПРЕДМЕТНОЙ ОБЛАСТИ.

Благодаря тактическому проектированию СОБЫТИЯ ПРЕДМЕТНОЙ ОБЛАСТИ становятся частью вашей модели предметной области и могут публиковаться и рассылаться в вашем ОГРАНИЧЕННОМ КОНТЕКСТЕ. Это очень удобный способ сообщать заинтересованным слушателям о важных событиях. Теперь вы узнаете, как моделировать СОБЫТИЯ ПРЕДМЕТНОЙ ОБЛАСТИ и использовать их в своих ОГРАНИЧЕННЫХ КОНТЕКСТАХ.

Проектирование, реализация и использование СОБЫТИЙ ПРЕДМЕТНОЙ ОБЛАСТИ

Для эффективного проектирования и реализации СОБЫТИЙ ПРЕДМЕТНОЙ ОБЛАСТИ в вашем ОГРАНИЧЕННОМ КОНТЕКСТЕ необходимо выполнить действия, описанные ниже. Кроме того, ниже приведены примеры того, как используются СОБЫТИЯ ПРЕДМЕТНОЙ ОБЛАСТИ.

```
public interface DomainEvent
{
  public Date OccurredOn
  {
    get;
  }
}
```

Этот код на языке C# можно было бы счесть минимальным интерфейсом, который должно обеспечивать каждое СОБЫТИЕ ПРЕДМЕТНОЙ ОБЛАСТИ. В частности, можно передать дату и время, когда произошло СОБЫТИЕ ПРЕДМЕТНОЙ ОБЛАСТИ, поэтому следует предусмотреть свойство `OccurredOn`. Эта деталь не является абсолютной необходимостью, но часто оказывается полезной. Таким образом, ваши типы СОБЫТИЙ ПРЕДМЕТНОЙ ОБЛАСТИ, вероятно, должны обеспечивать этот интерфейс.

Следует тщательно продумать имена типов СОБЫТИЙ ПРЕДМЕТНОЙ ОБЛАСТИ. Слова, которые вы используете, должны отражать ЕДИНЫЙ ЯЗЫК вашей модели. Они формируют мост между событиями в вашей модели и внешним миром. Крайне важно, чтобы вы правильно называли свои события.

Имена типов СОБЫТИЙ ПРЕДМЕТНОЙ ОБЛАСТИ должны быть утверждением о произошедшем, т.е. быть глаголом в прошедшем времени. Вот некоторые примеры из Контекста управления гибким проектированием: имя `ProductCreated`, например, утверждает, что продукт Scrum был создан в какой-то момент в прошлом. Другие СОБЫТИЯ ПРЕДМЕТНОЙ ОБЛАСТИ могут называться `ReleaseScheduled`, `SprintScheduled`, `BacklogItemPlanned` и `BacklogItemCommitted`. Каждое из этих имен ясно и кратко сообщает о том, что случилось в вашем СМЫСЛОВОМ ЯДРЕ.

Комбинация имен СОБЫТИЙ ПРЕДМЕТНОЙ ОБЛАСТИ и их свойств должна полностью описывать то, что случилось в модели предметной области. Но какую именно информацию должны содержать свойства СОБЫТИЙ ПРЕДМЕТНОЙ ОБЛАСТИ?

Спросите себя: что является причиной изданного СОБЫТИЯ ПРЕДМЕТНОЙ ОБЛАСТИ? Например, у события `ProductCreated` есть команда, которая его вызывает (команда — это объектная форма метода или запроса на выполнение действия). Команда называется `CreateProduct`. Таким образом, можно сказать, что событие `ProductCreated` является результатом команды `CreateProduct`.

Команда `CreateProduct` имеет следующие свойства: 1) `tenantId`, идентифицирующее арендатора-подписчика; 2) `productId`, идентифицирующее уникальный создаваемый объект класса `Product`; 3) имя `name` объекта класса `Product`; и 4) описание `description` объекта класса `Product`. *Каждое из этих свойств является необходимым для создания объекта класса* `Product`.

Следовательно, СОБЫТИЕ ПРЕДМЕТНОЙ ОБЛАСТИ ProductCreated должно содержать все свойства, обеспечиваемые командой, которая стала причиной создания события: 1) tenantId; 2) productId; 3) name и 4) description. Событие полностью и точно сообщит всем подписчикам, что случилось в модели, т.е. объект класса Product был создан для арендатора с идентификатором tenantId, имеет уникальный идентификатор productId, имя name и описание description.

Эти пять примеров дают хорошее представление о свойствах, которые необходимо включать в различные СОБЫТИЯ ПРЕДМЕТНОЙ ОБЛАСТИ, изданные Контекстом управления гибким проектированием. Например, когда объект класса BacklogItem передается объекту класса Sprint, публикуется СОБЫТИЕ ПРЕДМЕТНОЙ ОБЛАСТИ BacklogItemCommitted. Это СОБЫТИЕ ПРЕДМЕТНОЙ ОБЛАСТИ содержит свойства tenantId, свойство backlogItemId объекта BacklogItem, который был назначен для спринта, и идентификатор sprintId объекта класса Sprint, которому было передано событие.

Как описано в главе 4, иногда СОБЫТИЕ ПРЕДМЕТНОЙ ОБЛАСТИ можно обогатить дополнительными данными. Это может быть особенно полезным для пользователей, которые не хотят делать запросы к вашему ОГРАНИЧЕННОМУ КОНТЕКСТУ для получения дополнительных данных, которые им нужны. Даже в этом случае не следует заполнять СОБЫТИЯ ПРЕДМЕТНОЙ ОБЛАСТИ настолько большими данными, поскольку это приведет к потере смысла. Например, представьте себе, что объект класса BacklogItemCommitted хранит полное состояние объекта класса BacklogItem. Можно ли понять, что случилось, по такому СОБЫТИЮ ПРЕДМЕТНОЙ ОБЛАСТИ? Дополнительные

данные могут сделать это неясным, если не потребовать, чтобы пользователи глубоко разбирались в содержании элемента BacklogItem. Кроме того, представьте себе, что элемент класса BacklogItemUpdated содержит полное состояние объекта класса BacklogItem, кроме свойства BacklogItemCommitted. В этом случае совершенно неясно, что случилось с объектом класса BacklogItem, потому что для этого пользователь должен был бы сравнивать последний объект класса BacklogItemUpdated с предыдущим.

Для того чтобы разъяснить правильное использование СОБЫТИЙ ПРЕДМЕТНОЙ ОБЛАСТИ, рассмотрим сценарий, в котором владелец продукта передает объект класса BacklogItem объекту класса Sprint. Команда сама загружает объекты классов BacklogItem и Sprint, а затем применяется к АГРЕГАТУ BacklogItem. Это приводит к изменению состояния объекта класса BacklogItem и в результате публикуется СОБЫТИЕ ПРЕДМЕТНОЙ ОБЛАСТИ BacklogItemCommitted.

Важно, что измененные АГРЕГАТ и СОБЫТИЕ ПРЕДМЕТНОЙ ОБЛАСТИ будут храниться вместе в одной и той же транзакции. Если бы вы использовали инструмент объектно-реляционного отображения, то хранили бы АГРЕГАТ в одной таблице, СОБЫТИЕ ПРЕДМЕТНОЙ ОБЛАСТИ — в таблице хранилища событий, а затем зафиксировали бы транзакцию. Если бы вы использовали ИСТОЧНИКИ СОБЫТИЙ, то состояние АГРЕГАТА полностью представлялось бы непосредственно СОБЫТИЯМИ ПРЕДМЕТНОЙ ОБЛАСТИ. (ИСТОЧНИКИ СОБЫТИЙ рас-

сматриваются в следующем разделе этой главы.) В любом случае хранение СОБЫТИЙ ПРЕДМЕТНОЙ ОБЛАСТИ в хранилище сохраняет причинную согласованность модели предметной области.

Как только СОБЫТИЕ ПРЕДМЕТНОЙ ОБЛАСТИ будет сохранено в хранилище событий, оно может быть разослано любым заинтересованным сторонам. Они могут находиться как в пределах вашего собственного ОГРАНИЧЕННОГО КОНТЕКСТА, так и во внешних ОГРАНИЧЕННЫХ КОНТЕКСТАХ. Это ваш способ сообщить миру о том, что в вашем СМЫСЛОВОМ ЯДРЕ произошло нечто примечательное.

Отметим, что один лишь причинный порядок СОБЫТИЙ ПРЕДМЕТНОЙ ОБЛАСТИ не гарантирует его сохранения в других распределенных узлах. Таким образом, распознавание правильной причинной связи возлагается на ОГРАНИЧЕННЫЙ КОНТЕКСТ пользователя. Причинная связь может быть установлена самим типом СОБЫТИЙ ПРЕДМЕТНОЙ ОБЛАСТИ или метаданными, связанными с СОБЫТИЕМ ПРЕДМЕТНОЙ ОБЛАСТИ, например, идентификатором последовательности или причинности, который указывал бы на то, что вызвало данное СОБЫТИЕ ПРЕДМЕТНОЙ ОБЛАСТИ, и если причина еще не была замечена, то пользователь должен отложить использование поступившего события, пока не будет получена его причина. В некоторых случаях можно игнорировать скрытые СОБЫТИЯ ПРЕДМЕТНОЙ ОБЛАСТИ, которые уже были отложены; в этом случае причинную связь можно устранить.

Отметим еще одно важное обстоятельство. Хотя обычно эта команда генерируется пользовательским интерфейсом, иногда СОБЫТИЯ ПРЕДМЕТНОЙ ОБЛАСТИ могут быть вызваны другим источником. Это может быть таймер, фиксирующий конец периода, например, конец рабочего дня, недели, месяца или года. В таких ситуациях событие не является следствием выполнения команды, потому что конец периода времени — это факт, и если бизнес следит за этим фактом, то конец периода моделируется как СОБЫТИЕ ПРЕДМЕТНОЙ ОБЛАСТИ, а не как команда.

Такое событие, как конец истекающего периода времени, должно иметь подробное имя, которое является частью ЕДИНОГО ЯЗЫКА. Например, "Конец финансового года" может быть важным событием, на который ваш бизнес должен реагировать. Кроме того, событие `4:00 p.m.` (`16:00`) на Уолл-Стрит известно как "Закрытие рынков", а не просто `4:00 p.m.` Следовательно, необходимо иметь имя для специфического хронологического СОБЫТИЯ ПРЕДМЕТНОЙ ОБЛАСТИ.

Команда отличается от СОБЫТИЙ ПРЕДМЕТНОЙ ОБЛАСТИ тем, что в некоторых случаях команда может быть отклонена как несоответствующая, например, в результате недоступности некоторых ресурсов (продуктов, капитала и т.д.) или проверки бизнес-правил. Таким образом, команда может быть отклонена, но хронологическое СОБЫТИЕ ПРЕДМЕТНОЙ ОБЛАСТИ не может быть отклонено. Даже в этом случае в ответ на хронологическое СОБЫТИЕ ПРЕДМЕТНОЙ ОБЛАСТИ приложение должно генерировать одну или несколько команд, чтобы выполнить некоторый набор действий.

ИСТОЧНИКИ СОБЫТИЙ

Шаблон ИСТОЧНИКИ СОБЫТИЙ можно описать как хранение всех СОБЫТИЙ ПРЕДМЕТНОЙ ОБЛАСТИ, связанных с экземпляром АГРЕГАТА, в виде записи того, что изменилось в АГРЕГАТЕ. Вместо того чтобы сохранять все состояние АГРЕГАТА, в памяти хранятся все произошедшие СОБЫТИЯ ПРЕДМЕТНОЙ ОБЛАСТИ, которые связаны с ним. Давайте посмотрим, как это делается.

Все СОБЫТИЯ ПРЕДМЕТНОЙ ОБЛАСТИ, которые произошли с одним экземпляром АГРЕГАТА, упорядочиваются, поскольку они образуют поток событий. Поток событий начинается с первого СОБЫТИЯ ПРЕДМЕТНОЙ ОБЛАСТИ, который когда-либо происходил с экземпляром АГРЕГАТА, и продолжается до последнего произошедшего СОБЫТИЯ ПРЕДМЕТНОЙ ОБЛАСТИ. Когда происходят новые СОБЫТИЯ ПРЕДМЕТНОЙ ОБЛАСТИ с данным экземпляром АГРЕГАТА, они добавляются в конец его потока событий. Повторное воспроизведение потока событий, связанных с АГРЕГАТОМ, позволяет воссоздать состояние по информации, полученной из хранилища, и записать ее обратно в память. Иначе говоря, с помощью ИСТОЧНИКА СОБЫТИЙ АГРЕГАТ, который был удален из памяти по какой-то причине, можно полностью воссоздать на основе его потока событий.

В предыдущей диаграмме первым произошедшим СОБЫТИЕМ ПРЕДМЕТНОЙ ОБЛАСТИ был экземпляр класса BacklogItemPlanned; следующим — BacklogItemStoryDefined, а последним — BacklogItemCommitted. Полный поток событий в данный момент состоит из этих трех событий, и они следуют в порядке, описанном на диаграмме.

Каждое из СОБЫТИЙ ПРЕДМЕТНОЙ ОБЛАСТИ, которое в данном примере происходит с АГРЕГАТОМ, было вызвано командой. В примере, показанном на диаграмме, этой командой является CommitBacklogItemToSprint, которая вызвала СОБЫТИЕ ПРЕДМЕТНОЙ ОБЛАСТИ BacklogItemCommitted.

Stream Id	Stream Version	Event Type	Event Content
backlogItem123	1	BacklogItemPlanned	{ }
backlogItem123	2	BacklogItemStoryDefined	{ }
backlogItem123	3	BacklogItemCommitted	{ }
	N		{ }
	N		{ }
	N		{ }

Хранилище событий — это последовательная коллекция или таблица, в которую добавляются все СОБЫТИЯ ПРЕДМЕТНОЙ ОБЛАСТИ. Поскольку хранилище событий работает в режиме, допускающем только добавление (append only), механизм хранения работает чрезвычайно быстро. Это позволяет планировать СМЫСЛОВОЕ ЯДРО, использующее ИСТОЧНИКИ СОБЫТИЙ, чтобы обеспечить очень высокую производительность, малое время ожидания и возможность высокой масштабируемости.

Производительность

Если одной из основных целей вашей работы является производительность, то вы оцените информацию о кешировании и снимках. Прежде всего, в памяти кешируются высокопроизводительные АГРЕГАТЫ, что позволяет не воссоздавать их каждый раз по информации, хранящейся в памяти. Использование модели АКТОРА, в которой акторами являются АГРЕГАТЫ [Reactive], — один из наиболее простых способов кеширования состояния АГРЕГАТОВ.

Другой инструмент, имеющийся в вашем распоряжении, — это снимки (snapshots), благодаря которым АГРЕГАТЫ, которые были выгружены из памяти, можно оптимальным образом воссоздать, не перезагружая каждое СОБЫТИЕ ПРЕДМЕТНОЙ ОБЛАСТИ из потока событий. Этот метод сводится к поддержанию снимков некоторых последовательных состояний вашего АГРЕГАТА (объекта, актора или записи) в базе данных. Снимки более подробно обсуждаются в книгах *Implementing Domain-Driven Design* [IDDD] и *Reactive Messaging Patterns with the Actor Model* [Reactive].

Одно из самых больших преимуществ использования ИСТОЧНИКОВ СОБЫТИЙ заключается в том, что они сохраняют запись обо всем, что когда-либо случалось в вашем СМЫСЛОВОМ ЯДРЕ, и позволяют различать отдельные события. Это может быть очень полезным для вашего бизнеса по многим причинам, как по тем, которые можно себе представить в настоящее время, например, связанным с обеспечением бизнес-правил и бизнес-анализом, так и по тем, которые вы пока не понимаете. Существуют также технические преимущества. Например, разработчики программного обеспечения могут использовать потоки событий для исследования тенденций использования и отладки исходного кода.

Полное описание методики ИСТОЧНИКИ СОБЫТИЙ можно найти в книге *Implementing Domain-Driven Design* [IDDD]. Кроме того, когда вы используете ИСТОЧНИКИ СОБЫТИЙ, вы почти наверняка обязаны использовать шаблон CQRS. Вы можете также найти обсуждение этой темы в книге *Implementing Domain-Driven Design* [IDDD].

Резюме

В этой главе вы узнали:

- как создавать и называть ваши СОБЫТИЯ ПРЕДМЕТНОЙ ОБЛАСТИ;
- что определение и реализация стандартного интерфейса ПРЕДМЕТНОЙ ОБЛАСТИ являются важными аспектами;
- что правильное название СОБЫТИЙ ПРЕДМЕТНОЙ ОБЛАСТИ является особенно важным;
- как определять свойства ваших СОБЫТИЙ ПРЕДМЕТНОЙ ОБЛАСТИ;
- что некоторые СОБЫТИЯ ПРЕДМЕТНОЙ ОБЛАСТИ могут быть вызваны командами, в то время как другие могут происходить из-за обнаружения некоторого состояния (например, даты или времени);
- как сохранять ваши СОБЫТИЯ ПРЕДМЕТНОЙ ОБЛАСТИ в хранилище событий;
- как публиковать СОБЫТИЯ ПРЕДМЕТНОЙ ОБЛАСТИ после того, как они были сохранены;
- что такое ИСТОЧНИКИ СОБЫТИЙ и как ваши СОБЫТИЯ ПРЕДМЕТНОЙ ОБЛАСТИ можно сохранять и использовать для представления состояния ваших АГРЕГАТОВ.

Полное описание СОБЫТИЙ ПРЕДМЕТНОЙ ОБЛАСТИ и вопросов интеграции см. в главах 8 и 13 книги *Implementing Domain-Driven Design* [IDDD].

Глава 7

Инструментальные средства для повышения эффективности проектирования

Используя DDD, необходимо глубоко разбираться в бизнес-процессах, а затем, основываясь на полученных знаниях, моделировать программное обеспечение. Это действительно процесс изучения, экспериментирования, обсуждения, более глубокого изучения и повторного моделирования. Мы должны углубить и лаконично изложить большой объем знаний и создать проект, эффективно удовлетворяющий стратегические потребности организации. Проблема заключается в том, что мы должны быстро учиться. В быстро изменяющейся промышленности мы обычно испытываем дефицит времени, потому что время имеет значение и, вообще, управляет многими нашими решениями, возможно, даже больше, чем следует. Если мы не успеваем вовремя и выходим за рамки сметы, то, независимо от того, какого уровня достигло наше программное обеспечение, мы, по-видимому, терпим неудачу. Все рассчитывают на нашу помощь, стремясь к успеху.

Некоторые специалисты предпринимали попытки убедить свое руководство в том, что большинство оценок времени, необходимого на выполнение проектов, бесполезны и не могут использоваться на практике. Я не уверен, что их усилия увенчались успехом, но каждый клиент, с которым я работал, всегда оказывал на меня давление, стремясь установить очень определенные временные рамки и ограничить по времени процесс проектирования

и реализации. В лучшем случае это приводит к постоянной борьбе между разработчиками программного обеспечения и руководством.

К сожалению, распространенным ответом на это негативное давление стало стремление экономить средства и сокращать календарные планы за счет проектирования. (В главе 1 мы доказали, что проектирование является обязательным, каким бы оно ни было — плохим или хорошим.) Итак, вы просто обязаны выполнять календарный план и вовремя заканчивать проектирование, используя подходы, которые помогут вам принимать хорошие проектные решения в заданных временных рамках.

В этой главе описываются полезные инструментальные средства для ускорения проектирования. Сначала мы обсудим СОБЫТИЙНЫЙ ШТУРМ, а затем рассмотрим способы улучшения артефактов, созданных в процессе разработки, чтобы получить осмысленные и, главное, реальные оценки календарного плана и сметы.

СОБЫТИЙНЫЙ ШТУРМ

СОБЫТИЙНЫЙ ШТУРМ (EVENT STORMING) — **это методика быстрого проектирования, предназначенная для вовлечения** ЭКСПЕРТОВ ПРЕДМЕТНОЙ ОБЛАСТИ **и разработчиков в быстро изменяющийся процесс изучения. Она сосредоточена на бизнесе и бизнес-процессах, а не на именах существительных и данных.**

До изучения СОБЫТИЙНОГО ШТУРМА я использовал методику, которую называл событийно-управляемым моделированием. Обычно она подразумевала беседы, конкретные сценарии и событийно-ориентированное моделирование с помощью очень упрощенного варианта языка UML. Диаграммы UML можно как рисовать на доске, так и воплощать в программах. Однако, как вы, вероятно, знаете, очень немногие бизнес-эксперты знают или используют хотя бы самый простой вариант UML. В результате в большинстве случаев моим единственным собеседником был другой разработчик,

понимающий основы UML. Это был очень полезный подход, но нужно было найти способ непосредственно вовлечь бизнес-экспертов в процесс проектирования. Вероятно, это означало отказ от UML в пользу более привлекательных для бизнес-экспертов инструментов проектирования.

Впервые я узнал о СОБЫТИЙНОМ ШТУРМЕ много лет назад от Альберто Брандолини (Alberto Brandolini) [Ziobrando], который также экспериментировал с другой формой событийно-управляемого моделирования. Однажды, испытывая острый недостаток времени, Альберто решил отказаться от UML и вместо него использовать стикеры. Так появился новый метод быстрого обучения и проектирования программного обеспечения, который позволял быстро вовлечь в процесс обсуждения всех его участников. Ниже перечислены некоторые из его преимуществ.

- Это очень осязательный подход. Каждый участник получает стопку стикеров и ручку и должен внести свой вклад в изучение и решение проблемы. И бизнес-эксперты, и разработчики участвуют в этом процессе на равных, поскольку они вместе изучают систему. Каждый делает свой вклад в ЕДИНЫЙ ЯЗЫК.
- Этот подход сосредоточивает внимание каждого на событиях и бизнес-процессе, а не на классах и базе данных.
- Это очень визуальный подход, который позволяет отказаться от экспериментального кодирования и вовлечь в процесс проектирования всех без исключения.
- СОБЫТИЙНЫЙ ШТУРМ можно организовать очень быстро и дешево. Вы можете буквально штурмовать новое СМЫСЛОВОЕ ЯДРО в течение часов вместо недель. Если вы напишете что-то на стикере, а позже решите, что это неправильно, то просто скомкаете и выбросите стикер. Эта ошибка будет стоить всего один-два цента, а значит, вносить изменения в проект станет проще. Ведь не каждый согласится на изменение проекта, если в него уже вложено много средств.
- Ваша группа совершит прорыв и получит глубокие знания. Точка. Это неизбежно. Некоторые придут на сеанс штурма, думая, что они достаточно хорошо понимают основную бизнес-модель, но после сеанса они будут лучше понимать бизнес-процесс и даже узнают нечто новое.
- Каждый получает какие-то знания. Являетесь ли вы ЭКСПЕРТОМ ПРЕДМЕТНОЙ ОБЛАСТИ или разработчиком программного обеспечения, после сеанса у вас будет четкое и ясное понимание модели. Это не значит, что вы обязательно сделаете открытие. Эти знания важны сами по себе. Во множестве проектов, по крайней мере, некоторые, а воз-

можно, и многие, участники не понимают, что делают, пока ошибка не будет реализована в виде кода. Модель, выработанная в результате СОБЫТИЙНОГО ШТУРМА, устраняет недоразумения и приводит всех к общему пониманию цели.

- Этот подход подразумевает идентификацию проблем, связанных с моделью и знаниями о бизнес-процессах, на самых ранних этапах. Устраняйте недоразумения и пользуйтесь результатами открытий. Выгоду получит каждый участник штурма.
- Вы можете использовать СОБЫТИЙНЫЙ ШТУРМ и для КОНЦЕПТУАЛЬНОГО, и для ПРОЕКТНОГО МОДЕЛИРОВАНИЯ. Выполнение концептуального штурма будет менее точным, в то время как штурм на уровне проекта приведет вас к созданию определенных артефактов программного обеспечения.
- Необязательно ограничиваться одним сеансом штурма. Вы можете провести двухчасовой сеанс штурма, а затем сделать перерыв. Зафиксируйте ваши достижения и вернитесь к ним на следующий день, чтобы потратить еще час или два на их усовершенствование. Если вы будете делать это в течение двух часов в день в течение трех или четырех дней, то достигнете глубокого понимания вашего СМЫСЛОВОГО ЯДРА и обеспечите интеграцию с окружающими ПОДОБЛАСТЯМИ.

Ниже приведен список всего необходимого для штурма модели.

- Необходимо привлечь к штурму нужных людей — ЭКСПЕРТОВ ПРЕДМЕТНОЙ ОБЛАСТИ и разработчиков, которые должны работать над моделью. У каждого из них будут свои вопросы и ответы. Для того чтобы поддерживать друг друга, они все должны находиться в одной комнате в течение всех сеансов моделирования.
- Каждый должен быть непредвзятым и свободным от предубеждений. Самая большая ошибка, которую люди часто совершают в ходе СОБЫТИЙНОГО ШТУРМА, — попытка слишком быстро достичь идеального решения. Скорее всего, вы создадите очень большое количество событий. Много событий — лучше, чем мало, потому что события лучше всего способствуют углублению знаний. У вас еще будет время уточнить проект, причем быстро и без больших затрат.
- Запаситесь большим количеством стикеров. Как минимум, вам понадобятся стикеры следующих цветов: оранжевого, фиолетового или красного, светло-голубого, бледно-желтого, сиреневого и розового. Могут пригодиться и другие цвета (например, зеленый; см. примеры, приведенные далее). Стикеры могут быть квадратными (3×3 дюйма, или 7,62×7,62 см) или прямоугольными. Вы не должны много

писать на стикерах: обычно на них пишут всего несколько слов. Постарайтесь, чтобы стикеры были достаточно липкими, если не хотите, чтобы они постоянно падали на пол.

- Дайте каждому человеку один черный маркер, который хорошо пишет. Лучше всего использовать для этого маркеры с тонким кончиком.
- Найдите широкую стену, на которой вы сможете осуществлять моделирование. Ширина более важна, чем высота, но высота поверхности для моделирования должна быть приблизительно один метр. Ширина должна быть фактически неограниченной, но, как минимум, следует предусмотреть 10 метров. Вместо стены для этого всегда можно использовать длинный стол для переговоров или пол. Правда, стол в конечном счете ограничит ваше пространство для моделирования, а пол для некоторых участников группы может оказаться неудобным. Стена является наилучшим вариантом.
- Возьмите длинный рулон бумаги, которую можно купить в магазинах художественных принадлежностей или канцелярских товаров. Бумага должна иметь размеры, указанные выше, т.е. не менее 10 метров в ширину и 1 метр в высоту. Приклейте бумагу на стену, используя клейкую ленту. Некоторые люди не хотят использовать бумагу, ограничиваясь только доской. Это может работать некоторое время, но стикеры со временем теряют липкость и падают с доски, особенно если они были переклеены и повторно прикреплялись в другом месте. На бумаге стикеры держатся дольше. Если вы намереваетесь заниматься моделированием в течение кратких промежутков времени на протяжении трех-четырех дней, а не в течение одного длинного сеанса, то долговечность стикеров имеет значение.

Имея достаточный объем ресурсов и пригласив компетентных людей для участия в сеансе, вы готовы начать. Рассмотрите каждый из следующих шагов один за другим.

1. *Штурм бизнес-процесса с помощью создания ряда* СОБЫТИЙ ПРЕДМЕТНОЙ ОБЛАСТИ *на стикерах. Самый популярный цвет для* СОБЫТИЙ ПРЕДМЕТНОЙ ОБЛАСТИ *— оранжевый.* Оранжевый цвет позволяет выделять СОБЫТИЯ ПРЕДМЕТНОЙ ОБЛАСТИ на поверхности моделирования.

Ниже перечислено несколько основных рекомендаций, которым вы должны следовать при создании СОБЫТИЙ ПРЕДМЕТНОЙ ОБЛАСТИ.

- Создание СОБЫТИЙ ПРЕДМЕТНОЙ ОБЛАСТИ в первую очередь должно подчеркнуть, что вы сосредоточены на бизнес-процессе, а не на данных и их структурах. Для этого вашей группе может понадобиться 10–15 минут, но не торопитесь и не забегайте вперед.
- Напишите на стикере имя каждого СОБЫТИЯ ПРЕДМЕТНОЙ ОБЛАСТИ. Как указывалось в предыдущей главе, имя должно быть глаголом в прошедшем времени. Например, одно событие можно назвать `ProductCreated`, а другое — `BacklogItemCommitted`. (Вы, конечно, можете переносить эти имена на несколько строчек, записывая их на стикерах.) Если вы осуществляете КОНЦЕПТУАЛЬНЫЙ ШТУРМ и считаете, что эти имена слишком конкретные, то используйте другие.
- Размещайте стикеры на вашей поверхности моделирования в хронологическом порядке, слева направо в том порядке, в котором каждое событие происходит в области. Начинайте с первых СОБЫТИЙ ПРЕДМЕТНОЙ ОБЛАСТИ на левом краю поверхности моделирования, а затем постепенно двигайтесь вправо. Если хронологический порядок событий неясен, тогда просто поместите соответствую-

щие СОБЫТИЯ ПРЕДМЕТНОЙ ОБЛАСТИ в какое-нибудь место модели. Впоследствии вы сможете выяснить, когда происходит событие, и уточнить модель.

- Одновременные СОБЫТИЯ ПРЕДМЕТНОЙ ОБЛАСТИ можно размещать одно под другим. Это подчеркнет, что их обработка должна происходить параллельно.
- Когда закончится определенная часть сеанса штурма, вы обнаружите проблемные точки в существующем или новом бизнес-процессе. Четко выделите их с помощью фиолетового или красного стикера и текста, объясняющего, в чем заключается проблема. Вы должны потратить время на такие точки, чтобы узнать больше.
- Иногда результатом СОБЫТИЯ ПРЕДМЕТНОЙ ОБЛАСТИ является ПРОЦЕСС, который необходимо запустить. Он может состоять из одного или многих шагов. Каждое СОБЫТИЕ ПРЕДМЕТНОЙ ОБЛАСТИ, которое запускает ПРОЦЕСС, должно быть зафиксировано и названо на сиреневом стикере. Проведите стрелку от СОБЫТИЯ ПРЕДМЕТНОЙ ОБЛАСТИ к названному ПРОЦЕССУ (сиреневому стикеру). Моделируйте мелкомодульные СОБЫТИЯ ПРЕДМЕТНОЙ ОБЛАСТИ, только если это важно для вашего СМЫСЛОВОГО ЯДРА. Например, регистрация пользователей, видимо, необходима, но все же она не является основной функцией вашего приложения. Моделируйте регистрационный процесс как одно отдельное крупное событие с именем `UserRegistered` и переходите далее. Сосредоточьте свои усилия на более важных событиях.

Если вы думаете, что исчерпали все возможные СОБЫТИЯ ПРЕДМЕТНОЙ ОБЛАСТИ, значит, пришло время отдохнуть, чтобы вернуться к моделированию позже. Вернувшись к моделированию через день, вы, без сомнения, найдете пропущенные концепции, а также уточните или удалите события, которые ранее считали важными. В какой-то момент вы обнаружите большую часть самых важных СОБЫТИЙ ПРЕДМЕТНОЙ ОБЛАСТИ. Тогда вы должны перейти на следующий этап.

2. *Создайте все* КОМАНДЫ, *вызывающие* СОБЫТИЯ ПРЕДМЕТНОЙ ОБЛАСТИ. Иногда СОБЫТИЕ ПРЕДМЕТНОЙ ОБЛАСТИ является результатом события, произошедшего в другой системе и поступившего в вашу систему. Однако часто КОМАНДА, вызывающая СОБЫТИЕ ПРЕДМЕТНОЙ ОБЛАСТИ, является результатом некоторого жеста пользователя. Имя КОМАНДЫ должно быть глаголом в повелительном наклонении, например `CreateProduct` и `CommitBacklogItem`. Ниже приведены основные рекомендации, касающиеся КОМАНД.

- Пишите имя КОМАНД, вызывающих соответствующие СОБЫТИЯ ПРЕДМЕТНОЙ ОБЛАСТИ, на светло-голубом стикере. Например, если у вас есть СОБЫТИЕ ПРЕДМЕТНОЙ ОБЛАСТИ по имени `BacklogItemCommitted`, то соответствующую КОМАНДУ, вызывающую это событие, назовите `CommitBacklogItem`.
- Размещайте светло-голубые стикеры КОМАНД слева от СОБЫТИЙ ПРЕДМЕТНОЙ ОБЛАСТИ, которые они вызывают. Они образуют пары: КОМАНДА/СОБЫТИЕ, КОМАНДА/СОБЫТИЕ, КОМАНДА/СОБЫТИЕ и т.д. Помните, что некоторые СОБЫТИЯ ПРЕДМЕТНОЙ ОБЛАСТИ происходят из-за достижения установленных сроков и поэтому, возможно, не имеют КОМАНД, которые их явно вызывают.
- Если есть определенная пользовательская роль, которая выполняет действие, это обстоятельство следует обязательно отметить, разместив маленький ярко-желтый стикер в левом нижнем углу

светло-голубой КОМАНДЫ, нарисовав на нем схематическую фигурку человека и название роли. Например, фигурка может символизировать роль "Владелец продукта", который выполняет КОМАНДУ.

- Иногда КОМАНДА запускает ПРОЦЕСС. Он может состоять их одного или многих шагов. Каждая КОМАНДА, запускающая ПРОЦЕСС, должна быть зафиксирована и названа на сиреневом стикере. Проведите стрелку от КОМАНДЫ к запускаемому ПРОЦЕССУ (сиреневый стикер). ПРОЦЕСС будет фактически вызывать одну или несколько КОМАНД и последующих СОБЫТИЙ ПРЕДМЕТНОЙ ОБЛАСТИ, и если вы их уже знаете, то отметьте их на стикерах и покажите, что они исходят из этого ПРОЦЕССА.
- Продолжайте перемещаться слева направо в хронологическом порядке, как вы это делали, создавая СОБЫТИЯ ПРЕДМЕТНОЙ ОБЛАСТИ.
- Возможно, создание КОМАНД заставит вас подумать о СОБЫТИЯХ ПРЕДМЕТНОЙ ОБЛАСТИ (например, при выявлении сиреневых и других ПРОЦЕССОВ) иначе, чем вы думали о них раньше. Используйте это открытие, разместив обнаруженное СОБЫТИЕ ПРЕДМЕТНОЙ ОБЛАСТИ на поверхности моделирования наряду с вызвавшей его КОМАНДОЙ.
- Вы можете также обнаружить, что одна КОМАНДА может вызывать несколько СОБЫТИЙ ПРЕДМЕТНОЙ ОБЛАСТИ. Это прекрасно; моделируйте одну КОМАНДУ и разместите ее слева от СОБЫТИЙ ПРЕДМЕТНОЙ ОБЛАСТИ, которые она вызывает.

Как только вы обнаружите все КОМАНДЫ, связанные с СОБЫТИЯМИ ПРЕДМЕТНОЙ ОБЛАСТИ, которые они вызывают, вы готовы перейти к следующему этапу.

3. *Установите связи между* СУЩНОСТЯМИ/АГРЕГАТАМИ, *к которым применяется команда, и* СУЩНОСТЯМИ/АГРЕГАТАМИ, *порождающими* СОБЫТИЯ ПРЕДМЕТНОЙ ОБЛАСТИ. Они представляют собой место хранения данных, в которых выполняются КОМАНДЫ и генерируются СОБЫТИЯ ПРЕДМЕТНОЙ ОБЛАСТИ. Создание диаграмм связей между сущностями часто является первым и наиболее популярным шагом в сегодняшнем мире информационных технологий, но большая ошибка начинать проектирование с этого шага. Бизнесмены их плохо понимают и могут быстро прекратить обсуждение. Этот шаг был перенесен на третье место в СОБЫТИЙНОМ ШТУРМЕ, потому что мы хотим в первую очередь сосредоточиться на бизнес-процессе, а не на данных. Если мы и должны подумать о данных в некоторый момент, то на этом этапе. На данном этапе бизнес-эксперты, вероятно, поймут, что в игру вступают данные. Ниже приводятся некоторые рекомендации, касающиеся моделирования АГРЕГАТОВ.

   - Если бизнес-эксперты не любят слово АГРЕГАТ или если оно их запутывает, вы должны использовать другое название. Обычно они могут понять термин СУЩНОСТЬ или просто ДАННЫЕ. Важно только,

чтобы это название не мешало группе иметь ясное представление о соответствующей концепции. Используйте для АГРЕГАТОВ светло-желтые стикеры, на которых написаны имена АГРЕГАТОВ. Это имя должно быть именем существительным, например Product или BacklogItem. Сделайте это для каждого АГРЕГАТА в вашей модели.

- Поместите стикер АГРЕГАТА немного позади и выше КОМАНДЫ и СОБЫТИЯ ПРЕДМЕТНОЙ ОБЛАСТИ, образующих пару. Иначе говоря, вы должны иметь возможность прочитать имя существительное, написанное на стикере АГРЕГАТА, но пара КОМАНДА/СОБЫТИЕ должна быть приклеена к нижней части стикера АГРЕГАТА, чтобы показать, что они связаны друг с другом. Если вы хотите оставить небольшой пробел между стикерами — пожалуйста, но только четко укажите, какая КОМАНДА и СОБЫТИЕ ПРЕДМЕТНОЙ ОБЛАСТИ какому АГРЕГАТУ принадлежат.
- По мере того как вы будете продвигаться вдоль временного графика бизнес-процесса, вы обнаружите, что АГРЕГАТЫ используются неоднократно. Не перестраивайте ваш временной график, чтобы переместить все пары КОМАНДА/СОБЫТИЕ под единственный стикер АГРЕГАТА. Просто напишите имя того же самого АГРЕГАТА на нескольких стикерах и разместите их на временном графике в тех местах, в которых возникают соответствующие пары КОМАНДА/СОБЫТИЕ. Главное — моделирование бизнес-процесса, а он протекает во времени.
- Возможно, что, размышляя о данных, связанных с различными действиями, вы обнаружите новые СОБЫТИЯ ПРЕДМЕТНОЙ ОБЛАСТИ. Не игнорируйте их. Просто поместите вновь обнаруженные СОБЫТИЯ ПРЕДМЕТНОЙ ОБЛАСТИ перед соответствующими КОМАНДАМИ и АГРЕГАТАМИ на поверхности моделирования. Вы можете также обнаружить, что некоторые из АГРЕГАТОВ являются слишком сложными, и их необходимо разделить на управляемый ПРОЦЕСС (сиреневый стикер). Не игнорируйте эти возможности.

Закончив эту часть стадии проектирования, вы приблизитесь к дополнительным этапам, которые выполнять необязательно. Кроме того, если вы выбрали архитектуру ИСТОЧНИКИ СОБЫТИЙ, описанную в предыдущей главе, то уже проделали большой путь к пониманию вашего СМЫСЛОВОГО ЯДРА, потому что между СОБЫТИЙНЫМ ШТУРМОМ и ИСТОЧНИКАМИ СОБЫТИЙ есть много общего. Конечно, чем ближе ваш штурм к концептуальному, тем дальше вы находитесь от этапа фактической реализации. Однако вы можете использовать эту методику, чтобы получить представление на уровне проекта. По

моему опыту, в течение одного и того же сеанса штурма группы часто приближаются то к концептуальному моделированию, то к проектному. В конце концов, необходимость изучать определенные детали проекта заставит вас спуститься с уровня концептуальной модели на уровень проекта, который является основным.

4. *Нарисуйте границы и стрелки, чтобы показать поток событий на вашей поверхности моделирования.* Скорее всего, в результате СОБЫТИЙНОГО ШТУРМА вы обнаружите многочисленные модели и потоки СОБЫТИЙ ПРЕДМЕТНОЙ ОБЛАСТИ между ними. Ниже приводятся рекомендации по работе с ними.
   - Обычно границы возникают там, где существуют подразделения, в которых разные бизнес-эксперты дают несогласованные определения одного и того же термина, или существует важная концепция, не являющаяся частью СМЫСЛОВОГО ЯДРА.
   - С помощью черных маркеров можно нарисовать поверхность моделирования на бумаге. Покажите на ней контекст и другие границы. Используйте сплошные линии для ОГРАНИЧЕННЫХ КОНТЕКСТОВ и пунктирные линии для ПОДОБЛАСТЕЙ. *Очевидно, что границы на бумаге хранятся долго, поэтому сначала убедитесь, что вы понимаете все детали, прежде чем делать это. Если хотите отметить временные границы моделей, используйте розовые стикеры для общих областей и не торопитесь обводить их маркером, пока ваша уверенность не окрепнет.*
   - Разместите розовые стикеры в разных границах и укажите на них имена ОГРАНИЧЕННЫХ КОНТЕКСТОВ, соответствующих этим границам.
   - Нарисуйте стрелки, чтобы показать направление потоков СОБЫТИЙ ПРЕДМЕТНОЙ ОБЛАСТИ, существующих между ОГРАНИЧЕННЫМИ КОНТЕКСТАМИ. Это простой способ описания того, как некоторые

СОБЫТИЯ ПРЕДМЕТНОЙ ОБЛАСТИ **поступают в вашу систему, не будучи вызванными** КОМАНДАМИ **в вашем** ОГРАНИЧЕННОМ КОНТЕКСТЕ.

Любые другие детали, относящиеся к этим этапам, должны быть интуитивно понятны. Просто используйте для общения границы и линии.

5. *Идентифицируйте разные представления, которые ваши пользователи должны будут использовать для выполнения своих действий, а также важные роли разных пользователей.*
   - Вы не обязаны демонстрировать каждое представление, которое будет обеспечивать ваш пользовательский интерфейс, впрочем, как и любое другое представление. Представления, которые вы захотите показать, должны быть существенными и не требовать больших усилий для их создания. Представления можно отметить зелеными стикерами на поверхности моделирования. Если это поможет, нарисуйте примерный макет (или каркас) наиболее важных представлений пользовательского интерфейса.
   - С помощью ярко-желтых стикеров можно отметить важные роли пользователей. И снова напомним, что их следует демонстрировать, только если вы должны сообщить нечто важное о взаимодействии пользователя с системой, или что-то, что система делает для определенной роли.

Было бы неплохо, если бы четвертого и пятого дополнительных этапов оказалось достаточно для организации СОБЫТИЙНОГО ШТУРМА.

Другие инструменты

Конечно, все это не мешает вам экспериментировать, например, наносить на вашу поверхность моделирования другие рисунки и пытаться выполнить другие этапы в ходе вашего СОБЫТИЙНОГО ШТУРМА. Помните, что мы говорим об изучении и общении в ходе проектирования. Используйте любые средства, которые необходимы для моделирования. Только старайтесь избегать формальных и ненужных действий, которые будут стоить очень дорого. Ниже описаны некоторые альтернативные идеи.

Введите выполняемые спецификации высокого уровня, которые следуют подходу "если/когда/тогда". Они также известны как приемочные тесты. Вы можете больше прочитать об этом в книге Гойко Адзича (Gojko Adzic) *Specification by Example* [Specification]. Соответствующий пример приведен в главе 2. Только постарайтесь не перейти черту, за которой эти спецификации становятся самоцелью и получают приоритет над моделью предметной области. Я считаю, что для обслуживания и поддержки спецификаций требуется где-то на 15–25% больше времени и усилий по сравнению с обычным модульным тестированием (см. главу 2). При этом легко зациклиться на поддержании релевантности спецификаций текущему направлению бизнеса, поскольку модель со временем изменяется.

Примените методику, описанную в книге *Impact Mapping* [Impact Mapping], чтобы удостовериться, что программное обеспечение, которое вы проектируете, представляет СМЫСЛОВОЕ ЯДРО, а не другую менее важную модель. Эту методику также изобрел Гойко Аджич.

Прочитайте книгу Джефа Паттона (Jeff Patton) *User Story Mapping* [User Story Mapping]. Описанная в ней методика используется для концентрации внимания на СМЫСЛОВОМ ЯДРЕ, чтобы выяснить, на какие программные функции следует тратить время и средства.

Предыдущие три дополнительных инструмента имеют много общего с подходом DDD и были бы весьма уместны в любом проекте DDD. Все они предназначены для ускоренного и неформального проектирования без больших затрат.

Применение принципов DDD для гибкого проектирования

Я уже упоминал о движении *No Estimate*. Этот подход отрицает типичные методы, основанные на оценках, таких как очки за пользовательскую историю или количество операций в час. Он нацелен на повышение эффективности, а не контроль стоимости и не предусматривает оценивания любой задачи, для решения которой, вероятно, потребуется несколько

месяцев. Я не отклоняю этот подход. И все же, когда я писал книгу, клиенты, с которыми я работал, все еще требовали предоставлять сметы и оценки временных затрат даже для таких задач, как программирование мелкомодульных функций. Если безоценочный подход вас устраивает и не противоречит сложившейся ситуации, используйте его.

Я также знаю, что некоторые члены сообщества DDD определили собственные процессы и каркасы для выполнения процессов на основе принципов DDD и применяют их для проектирования. Этот подход может работать хорошо и эффективно, когда он принят конкретной группой, но намного труднее получить согласие на его использование от организаций, которые уже вложили капитал в каркасы гибкого проектирования, такие как Scrum.

Недавно на методику Scrum обрушилась волна критики. Хотя я не сторонник этой критики, я открыто заявлю, что часто или даже в большинстве случаев методика Scrum используется неправильно. Я уже упомянул о тенденции, когда группы сводят разработку к перетасовке задач на доске. Методика Scrum изобреталась не для этого. Повторюсь, *приобретение знаний* — это главный принцип Scrum и главная цель DDD, но она в значительной степени игнорируется и сводится к конвейерным разработкам на основе Scrum. Но даже несмотря на это, Scrum все еще очень широко используется в нашей промышленности, и я сомневаюсь, что в обозримом будущем эту методику заменит какая-то другая.

По этой причине я хочу продемонстрировать, как можно применить принципы DDD в проекте, создаваемом по методике Scrum. Описанные мною методы одинаково применимы к другим подходам к гибкому проектированию, таким как Kanban. Здесь нет ничего, что было бы присущие только методике Scrum, хотя часть рекомендаций сформулирована в терминах Scrum. Поскольку многие из читателей уже знакомы с этой методикой, внедряя ее в практику в той или иной форме, большинство моих рекомендаций будет относиться к модели предметной области и изучению, экспериментам и проектированию на основе принципов DDD. Общие рекомендации по

использованию Scrum, Kanban или другого подхода к гибкому проектированию вам придется искать в другом месте.

Там, где я использую термины *задача* или *доска задач*, они относятся ко всем методикам гибкого проектирования, даже к Kanban. Там, где я использую термин *спринт*, я буду пытаться вставлять слова *итерация*, если речь идет о гибком проектировании вообще, и *WIP* (work in progress — незавершенная работа), если речь идет о методике Kanban. Это может не всегда быть совершенно уместным, поскольку я не пытаюсь определять реальные процессы. Надеюсь, вы просто извлечете выгоду из идей и найдете способ правильно применить их в конкретном вашем каркасе гибкого проектирования.

Начнем сначала

Одно из самых важных условий, гарантирующих успешное использование подхода DDD для проектирования, состоит в том, чтобы нанять хороших специалистов. Хорошие и очень хорошие специалисты просто незаменимы. DDD — это продвинутая методология разработки программного обеспечения, требующая привлечения разработчиков высокого и очень высокого уровня. Никогда не недооценивайте важность найма правильных людей с правильными навыками и мотивацией.

Использование SWOT-анализа

Если вы не знакомы с основами SWOT-анализа [SWOT], то знайте, что его название представляет собой аббревиатуру от *Strengths* (Сильные сто-

роны), *Weaknesses* (Слабые стороны), *Opportunities* (Возможности) и *Threats* (Угрозы). SWOT-анализ — это способ размышлений о проекте очень определенным образом, позволяющим извлекать максимальный объем знаний за минимально возможное время. Ниже перечислены основные идеи, которые необходимо идентифицировать, работая на проектом.

- *Сильные стороны*: характеристики бизнеса или проекта, которые дают преимущество над другими.
- *Слабые стороны:* характеристики, ослабляющие бизнес или проект.
- *Возможности:* элементы, которые можно использовать в проекте, чтобы достичь преимущества.
- *Угрозы:* элементы внешней среды, которые могут осложнить ведение бизнеса или разработку проекта.

В любое время и в любом проекте, создаваемом по методике Scrum или с помощью любого другого метода гибкого проектирования, можно применить SWOT-анализ, чтобы выяснить текущую ситуацию.

1. Нарисуйте большую матрицу с четырьмя квадрантами.
2. Возьмите стикеры разного цвета для каждого квадранта SWOT.
3. Найдите сильные и слабые стороны вашего проекта, а также возможности и угрозы.
4. Напишите их на стикерах и разместите в соответствующих квадрантах матрицы.
5. Используйте SWOT-характеристики проекта (здесь особенно внимательно необходимо проанализировать модель предметной области), чтобы запланировать свои действия. На следующих этапах вы должны использовать сильные стороны и преодолеть слабости, которые могут оказывать очень большое влияние на ваш успех.

Как показано ниже, вы можете разместить эти действия на доске задач в ходе планирования проектных работ.

Всплески и долги моделирования

Вы впервые слышите термины *всплеск моделирования* (modeling spike) и *долг моделирования* (modeling debt) в проекте DDD?

Одна из лучших вещей, которые вы можете сделать в начале проекта, — организовать СОБЫТИЙНЫЙ ШТУРМ. Этот штурм и связанные с ним эксперименты создают всплеск моделирования. Вы должны приобрести знание о вашем Scrum-продукте, и платой за него будет всплеск. Однако я уже показал, как использование СОБЫТИЙНОГО ШТУРМА может значительно уменьшить стоимость необходимых инвестиций.

Очевидно, вы не можете ожидать, что с самого начала создадите идеальную модель предметной области, даже если в самом начале проекта организовали всплеск моделирования. Вы не достигнете цели, даже если проведете СОБЫТИЙНЫЙ ШТУРМ. Отчасти это объясняется тем, что бизнес и наше понимание бизнеса со временем изменяются, как и модель предметной области

Кроме того, если вы стремитесь ограничить во времени свои усилия по моделированию задачами на доске, то обязательно создадите небольшой долг моделирования в течение каждого спринта (или итерации, или WIP). У вас просто не будет времени для решения всех желательных задач. Например, вы можете начать работу и после экспериментирования понять, что существующий проект не соответствует бизнес-потребностям так, как вы ожидали. Однако календарный план будет требовать, чтобы вы двигались дальше.

Самое плохое, что вы можете сделать в такой ситуации, — просто забыть все, что вы узнали в результате моделирования, которое относилось к другому, более эффективному проекту. Вместо этого вы должны сделать примечание, что эти функции должно войти в более поздний спринт (или итерацию, или WIP). Этот пункт можно вставить в повестку дня ретроспективного совещания[1] и вернуть в виде новой задачи на следующем совещании по планированию спринта (или совещании по планированию итерации, или при следующем добавлении в очередь Kanban).

[1] В рамках подхода Kanban ретроспективные совещания могут проходить каждый день, поэтому вам не придется долго ждать возможности для улучшения модели.

Идентификация задач и оценивание

СОБЫТИЙНЫЙ ШТУРМ — это инструмент, который может использоваться в любое время, а не только в начале проектирования. В результате сеансов СОБЫТИЙНОГО ШТУРМА вы будете естественным образом создавать множество артефактов. Каждое из СОБЫТИЙ ПРЕДМЕТНОЙ ОБЛАСТИ, КОМАНДЫ и АГРЕГАТЫ, возникшие в бумажной модели, могут использоваться как единицы оценки. Как именно?

Тип компонента	Простой, ч	Умеренный, ч	Сложный, ч
СОБЫТИЕ ПРЕДМЕТНОЙ ОБЛАСТИ	0,1	0,2	0,3
КОМАНДА	0,1	0,2	0,3
АГРЕГАТ	1	2	4
...	...	...	...

Один из самых легких и точных способов оценивания — подход на основе показателей. Как видите, в этом случае создается простая таблица с единицами оценки для каждого типа компонентов модели, который вы должны будете реализовать. Это позволит удалить из оценок элементы угадывания и обеспечит научную основу для оценивания усилий. Вот как работает эта таблица.

1. Создаем один столбец для *Типа компонента*, чтобы описать определенный вид компонента, для которого определены единицы оценки.
2. Создаем три других столбца для *Простых*, *Умеренных* и *Сложных* компонентов. Эти столбцы будут содержать единицы оценки, выраженные в часах или долях часа.
3. Теперь создадим одну строку для каждого *Типа компонентов* в вашей архитектуре. В таблице показаны СОБЫТИЕ ПРЕДМЕТНОЙ ОБЛАСТИ,

КОМАНДА и АГРЕГАТЫ. Однако не ограничивайте себя ими. Создайте строку для других компонентов пользовательского интерфейса, службы, механизмов хранения, средств сериализации и десериализации СОБЫТИЙ ПРЕДМЕТНОЙ ОБЛАСТИ и т.д. Вы можете создавать строки для каждого вида артефакта, который создадите в исходном коде. (Если, например, обычно вы создаете средства сериализации и десериализации вместе с каждым СОБЫТИЕМ ПРЕДМЕТНОЙ ОБЛАСТИ, то назначьте оценку для СОБЫТИЙ ПРЕДМЕТНОЙ ОБЛАСТИ, отражающую сложность создания всех этих компонентов вместе взятых в каждом столбце.)

4. Укажите часы или доли часа, необходимые для каждого уровня сложности: простого, умеренного и сложного. Эти оценки включают не только время, необходимое для выполнения задачи, но и учитывают дальнейшие усилия по проектированию и тестированию. Стремитесь к точности оценок и будьте реалистами.
5. Зная незавершенные задачи (элементы бэклога, или WIP), поработайте над ними и получите точные значения показателей для каждой из этих задач. Для этого можно использовать электронную таблицу.
6. Добавьте все единицы оценки для всех компонентов в текущий спринт (итерацию или WIP), и они станут вашей полной оценкой.

Выполняя каждый спринт (итерацию или WIP), уточняйте показатели, чтобы знать, сколько именно часов или долей часа фактически требуется для выполнения работы. Если вы используете методологию Scrum и хотите перепроверить временные оценки, то поймете, что этот подход сильно упрощает оценки и значительно повышает их точность. По мере продвижения проекта оценки показателей будут становиться все более точными и реалистичными. Для этого может понадобиться всего несколько спринтов. Кроме того, оценки и опыт также изменяются со временем, так что показатели могут снизиться, а компоненты перейти из категории *Умеренный* в *Простой*.

Если вы используете подход Kanban и думаете, что эти оценки совершенно ошибочные и ненужные, задайте себе вопрос: "Как точно определить WIP, чтобы правильно ограничить очередь работ?" Независимо от того, что вы можете подумать, вы все еще оцениваете усилия и надеетесь, что ваши оценки являются правильными. Почему бы не добавить немного науки и использовать этот простой и точный подход к оцениванию?

К вопросу о точности

Этот подход работает. В одной большой корпоративной программе организация потребовала оценки для большого и сложного проекта в рамках широкой программы. Для решения этой задачи были назначены две

группы. Первой была группа, состоящая из высокооплачиваемых консультантов, которые работали с компаниями из списка Fortune 500 в области оценки и управления проектами. Они были бухгалтерами, имели докторские степени и были снабжены всем оборудованием, что обеспечивало им полное преимущество. Вторая группа архитекторов и разработчиков использовала процесс оценки на основе показателей. Стоимость проекта превышала 20 млн долл., и в итоге, когда обе оценки были получены, они различались всего на 200 тыс. долл. (группа технических специалистов получила более низкую оценку). Неплохо для техников.

Вы должны обеспечивать 20%-ную точность долгосрочных оценок и намного более точную оценку для более коротких сроков, например, для спринтов, итерации и очереди WIP.

Моделирование с ограничением времени

Теперь, имея оценки для каждого типа компонентов, вы можете оценивать ваши задачи непосредственно по этим компонентам. Вы можете сохранить все компоненты в виде отдельных задач, на решение которых тре-

буется определенное количество часов или долей часа, а также разделить задачи на более мелкие. Однако я предлагаю осторожно подходить к разделению задач на мелкие модули, чтобы не получить чрезмерно сложную доску задач. Как было показано ранее, может быть даже лучше объединить все КОМАНДЫ и все СОБЫТИЯ ПРЕДМЕТНОЙ ОБЛАСТИ, используемые отдельным АГРЕГАТОМ, в рамках отдельной задачи.

Реализация

Даже если вы имеете артефакты, идентифицированные в ходе СОБЫТИЙНОГО ШТУРМА, вы не обязательно обладаете полным знанием, необходимым для работы с определенным сценарием предметной области, пользовательской историей и сценарием использования. Если необходимы более глубокие знания, убедитесь, что в своих оценках вы учли время, необходимое для приобретения знаний. Время для чего? Напомню, что главе 2 было продемонстрировано создание конкретных сценариев на основе модели предметной области. Это может быть одним из лучших способов приобрести знание о вашем СМЫСЛОВОМ ЯДРЕ, помимо СОБЫТИЙНОГО ШТУРМА. Конкретные сценарии и СОБЫТИЙНЫЙ ШТУРМ — это два инструментальных средства, которые должны использоваться вместе. Вот как это работает.

- Выполните быстрый сеанс СОБЫТИЙНОГО ШТУРМА в течение часа или около того. Вы почти наверняка обнаружите, что нуждаетесь в разработке более конкретных сценариев на основе открытий, сделанных в ходе вашего моделирования.
- Объединитесь с ЭКСПЕРТОМ ПРЕДМЕТНОЙ ОБЛАСТИ, чтобы обсудить один или несколько конкретных сценариев, которые должны быть усовершенствованы. Это позволит выяснить, какая модель программного обеспечения будет использоваться. Снова напомню, что цель идентификации фактических элементов модели предметной области (напри-

мер, объектов) состоит в том, чтобы выяснить, как элементы взаимодействуют друг с другом и с пользователями (см. главу 2).

- Создайте ряд приемочных тестов (или выполняемых спецификаций) и выполните каждый из сценариев (см. главу 2).
- Создайте компоненты, чтобы выполнить тесты/спецификации. Выполните итерации (коротко и быстро) и уточняйте тесты/спецификации и компоненты, пока не достигнете того, чего ожидает ваш ЭКСПЕРТ ПРЕДМЕТНОЙ ОБЛАСТИ.
- Очень вероятно, что часть итерации (резюме и быстрый) заставит вас рассмотреть другие сценарии, создать дополнительные тесты/спецификации и усовершенствовать существующие и вновь созданные компоненты.

Продолжайте, пока не приобретете все знания, необходимые для достижения ограниченной бизнес-цели, или пока не истечет выделенное для этого время. Если вы не достигли желаемого результата, зафиксируйте долг моделирования, чтобы вернуться к нему в будущем (в идеале — ближайшем).

И все же, как долго вам будут нужны ЭКСПЕРТЫ ПРЕДМЕТНОЙ ОБЛАСТИ?

Взаимодействие с ЭКСПЕРТАМИ ПРЕДМЕТНОЙ ОБЛАСТИ

Одна из главных проблем, связанных с использованием DDD, — найти время для работы с ЭКСПЕРТАМИ ПРЕДМЕТНОЙ ОБЛАСТИ и не переусердствовать при этом. Во многих случаях ЭКСПЕРТЫ ПРЕДМЕТНОЙ ОБЛАСТИ имеют другие обязанности, встречи или могут быть в командировках. В результате им может быть трудно найти достаточно времени для разговора с вами. Итак, мы должны найти время и ограничиться только самым необходимым. Если вы не сделаете сеансы моделирования интересными и эффективными, то можете не получить нужный совет в нужное время. Если же ЭКСПЕРТЫ ПРЕДМЕТНОЙ ОБЛАСТИ сочтут общение с вами ценным и полезным, то вы, вероятно, установите крепкие деловые связи, которые вам необходимы.

Таким образом, первые вопросы, на которые необходимо ответить: когда нам нужны ЭКСПЕРТЫ ПРЕДМЕТНОЙ ОБЛАСТИ? Какие задачи они должны помочь нам выполнить?

- Всегда привлекайте ЭКСПЕРТОВ ПРЕДМЕТНОЙ ОБЛАСТИ к СОБЫТИЙНОМУ ШТУРМУ. У разработчиков будет много вопросов, а у ЭКСПЕРТОВ ПРЕДМЕТНОЙ ОБЛАСТИ — много ответов. Удостоверьтесь, что они вместе участвуют в СОБЫТИЙНОМ ШТУРМЕ.
- Вам понадобится участие ЭКСПЕРТОВ ПРЕДМЕТНОЙ ОБЛАСТИ в обсуждениях и разработке модельных сценариев (см. примеры в главе 2).
- ЭКСПЕРТЫ ПРЕДМЕТНОЙ ОБЛАСТИ необходимы для обзора тестов, предназначенных для проверки правильности модели. Это предполагает, что разработчики уже приняли сознательное решение придерживаться ЕДИНОГО ЯЗЫКА и используют качественные и реалистичные тестовые данные.
- Вам потребуются эксперты предметной области для уточнения ЕДИНОГО ЯЗЫКА и имен АГРЕГАТОВ, КОМАНД и СОБЫТИЙ ПРЕДМЕТНОЙ ОБЛАСТИ, определенных всей группой. Неоднозначности устраняются с помощью обзоров, вопросов и обсуждения. Даже в этом случае в сеансах СОБЫТИЙНОГО ШТУРМА должно быть решено большинство вопросов, связанных с ЕДИНЫМ ЯЗЫКОМ.

Итак, теперь, когда вы знаете, зачем нужны ЭКСПЕРТЫ ПРЕДМЕТНОЙ ОБЛАСТИ, сколько времени вы должны попросить у них для выполнения каждой из этих обязанностей?

- Сеансы СОБЫТИЙНОГО ШТУРМА должны быть ограничены несколькими часами (два или три каждый). Возможно, сеансы следует поочередно проводить в течение трех-четырех дней.

- Выделите достаточно времени для обсуждения и уточнения сценариев, но попытайтесь предусмотреть время для каждого из них. Вы должны обсудить и рассмотреть один сценарий в течение 10–20 минут.
- Вам потребуется определенный объем времени для работы над тестами вместе с ЭКСПЕРТАМИ ПРЕДМЕТНОЙ ОБЛАСТИ, чтобы сделать обзор того, что вы написали. Но не ожидайте, что эксперты сядут вместе с вами писать код. Возможно, они сделают это, и тогда это будет для вас приятной неожиданностью, но особо не рассчитывайте на это. Точные модели требуют меньше времени для изучения и проверки. Не следует недооценивать способность ЭКСПЕРТОВ ПРЕДМЕТНОЙ ОБЛАСТИ быстро прочитать тест с вашей помощью. Они могут сделать это, особенно если тестовые данные реалистичны. Ваши тесты должны позволить ЭКСПЕРТУ ПРЕДМЕТНОЙ ОБЛАСТИ читать, понимать и проверять один тест в течение примерно двух минут.
- В ходе обзора тестов ЭКСПЕРТЫ ПРЕДМЕТНОЙ ОБЛАСТИ могут предоставить входные данные для АГРЕГАТОВ, КОМАНД и СОБЫТИЙ ПРЕДМЕТНОЙ ОБЛАСТИ, а также, возможно, другие артефакты, согласованные с ЕДИНЫМ ЯЗЫКОМ. Этого можно достигнуть за короткое время.

Эти рекомендации должны помочь вам использовать только правильные объемы времени для работы с ЭКСПЕРТАМИ ПРЕДМЕТНОЙ ОБЛАСТИ и ограничивать время, которое вы проведете с ними.

Резюме

В этой главе вы узнали:

- о СОБЫТИЙНОМ ШТУРМЕ и способах его проведения вместе с вашей группой для ускорения процесса моделирования;
- о других инструментах, которые могут использоваться наряду с СОБЫТИЙНЫМ ШТУРМОМ;
- как использовать принципы DDD в ходе работы над проектом и как управлять оценками и временем, необходимым для общения с ЭКСПЕРТАМИ ПРЕДМЕТНОЙ ОБЛАСТИ.

Исчерпывающую справочную информацию о применении принципов DDD для проектирования вы найдете в книге *Implementing Domain-Driven Design* [IDDD].

Библиография

[BDD] North, Dan. "Behavior-Driven Development." 2006. http://dannorth.net/introducing-bdd/.

[Causal] Lloyd, Wyatt, Michael J. Freedman, Michael Kaminsky, and David G. Andersen. "Don't Settle for Eventual Consistency: Stronger Properties for Low-Latency Geo-replicated Storage." http://queue.acm.org/detail.cfm?id=2610533.

[DDD] Evans, Eric. Domain-Driven Design: Tackling Complexity in the Heart of Software. Boston: Addison-Wesley, 2004. (Эрик Эванс. *Предметно-ориентированное проектирование (DDD): структуризация сложных программных систем.* Пер. с англ., ИД "Вильямс", 2014 г., ISBN 978-5-8459-1942-7.)

[Essential Scrum] Rubin, Kenneth S. Essential Scrum: A Practical Guide to the Most Popular Agile Process. Boston: Addison-Wesley, 2012.

[IDDD] Vernon, Vaughn. Implementing Domain-Driven Design. Boston: Addison-Wesley, 2013. (Вон Вернон. *Реализация методов предметно-ориентированного проектирования.* Пер. с англ., ИД «Вильямс», 2016 г., ISBN 978-5-8459-1881-9.)

[Impact Mapping] Adzic, Gojko. Impact Mapping: Making a Big Impact with Software Products and Projects. Provoking Thoughts, 2012.

[Microservices] Newman, Sam. Building Microservices. Sebastopol, CA: O'Reilly Media, 2015.

[Reactive] Vernon, Vaughn. Reactive Messaging Patterns with the Actor Model: Applications and Integration in Scala and Akka. Boston: Addison-Wesley, 2015.

[RiP] Webber, Jim, Savas Parastatidis, and Ian Robinson. REST in Practice: Hypermedia and Systems Architecture. Sebastopol, CA: O'Reilly Media, 2010.

[Specification] Adzic, Gojko. Specification by Example: How Successful Teams Deliver the Right Software. Manning Publications, 2011.

[SRP] Wikipedia. "Single Responsibility Principle." http://en.wikipedia.org/wiki/Single_responsibility_principle.

[SWOT] Wikipedia. "SWOT Analysis." https://en.wikipedia.org/wiki/SWOT_analysis.

[User Story Mapping] Patton, Jeff. User Story Mapping: Discover the Whole Story, Build the Right Product. Sebastopol, CA: O'Reilly Media, 2014.

[WSJ] Andreessen, Marc. "Why Software Is Eating the World." Wall Street Journal, August 20, 2011.

[Ziobrando] Brandolini, Alberto. "Introducing EventStorming." https://leanpub.com/introducing_eventstorming.

Предметный указатель

А

Агрегат 25
Адаптер
  ввода 58
  вывода 58
Анализ SWOT 140
Архитектура
  CQRS 59
  REST 59
  Источники событий 58
  Микрослужбы 59
  Модель актора 59
  Облачные вычисления 59
  Порты и адаптеры 57
  Реактивная модель 59
  Сервис-ориентированная 59

Б

Большок ком грязи 33
Бэклог 19

Е

Единый язык 24

И

Издатель–Подписчик 76
Интерфейс
  RESTful 76
  рассылки сообщений 76
Итерация 140
Итоговая согласованность 54

К

Карта контекстов 24
Команда 132
Командное сообщение 83
Концептуальный штурм 130
Корневая сущность 91

М

Методология
  Kanban 139
  Scrum 22
Модель
  Актор 92
Модель предметной области 23
  анемичная 102
Модуль 65

Н

Незавершенная работа 140

О

Объект-значение 90
Ограниченный контекст 23, 28
Отношение
  Большой комок грязи 74
  Клиент-поставщик 71
  Конформист 71
  Общедоступный язык 73
  Общее ядро 70
  Отдельное существование 71, 74
  Партнерство 70
  Предохранительный уровень 72
  Служба с открытым протоколом 73

П

Подобласть 24, 62
  вспомогательная 63
  универсальная 63
Прикладная служба 92

Принцип
  единственной обязанности, SPR 97
Приобретение знаний 22
Программирование
  объектно-ориентированное 103
  функциональное 103
Проектирование
  стратегическое 23
  тактическое 24
Пространство
  задач 28
  решений 28
Протокол
  RESTful HTTP 78
  SOAP 77
Процесс 131

Р

Рассылка сообщений 80
  Доставка не реже одного раза 84
  Идемпотентный получатель 84

С

Связывание контекстов 24, 68
Служба приложения 58
Смысловое ядро 28, 62
Снимок 122
Событие предметной области 25
Событийный штурм 51, 126
Согласованность
  итоговая 102
  причинная 113
Сущность 90

У

Удаленный вызов процедур 76
Унаследованная система 64
  неограниченная 64

Э

Эксперт предметной области 24

АНАЛИЗ И ПРОЕКТИРОВАНИЕ ИНФОРМАЦИОННЫХ СИСТЕМ С ПОМОЩЬЮ UML 2.0

ТРЕТЬЕ ИЗДАНИЕ

Лешек А. Мацяшек

www.williamspublishing.com

Книга представляет собой новое издание популярного учебника Лешека Мацяшека по объектно-ориентированной разработке информационных систем. В книге подробно описаны методы анализа и проектирования промышленных информационных систем с использованием языка UML. Отличительной особенностью книги является обилие учебных примеров, упражнений, контрольных вопросов и многовариантных тестов. Уникальный характер книги обусловлен оптимальным сочетанием практического опыта и теоретических представлений. Книга будет полезна системным аналитикам и архитекторам, программистам, преподавателям и студентам высших учебных заведений, а также все специалистам по информационным технологиям.

ISBN 978-5-8459-1430-9 **в продаже**

РЕАЛИЗАЦИЯ МЕТОДОВ ПРЕДМЕТНО-ОРИЕНТИРОВАННОГО ПРОЕКТИРОВАНИЯ

Вон Вернон

www.williamspublishing.com

В книге описаны реализация методов предметно-ориентированного проектирования (DDD) и специализированные подходы к реализации систем на основе современной архитектуры, а также показана важность ориентации на предметную область с учетом технических ограничений. Автор описывает методы DDD на реалистичных примерах, написанных на языке Java. Все примеры объединены в рамках единого сценария: разработки системы управления гибким проектированием SaaS для многоарендной среды на основе методологии Scrum. Книга представляет собой ценный источник знаний по методам предметно-ориентированного проектирования и предназначена для программистов всех уровней.

ISBN 978-5-8459-1881-9 **в продаже**

АДАПТИВНЫЙ КОД НА C#
Проектирование классов и интерфейсов, шаблоны и принципы SOLID

Гэри Маклин Холл

www.williamspublishing.com

В этой книге рассматриваются практические вопросы гибкой разработки адаптивного кода с помощью проектных шаблонов и принципов SOLID: единственной ответственности, открытости-закрытости, подстановки Лисков, разделения интерфейса, внедрения зависимостей. В ней рассматривается организация процесса гибкой разработки приложений на C# по методике Scrum, выявление зависимостей и эффективного управления ними, программирование интерфейсов, применение шаблонов и исключение антишаблонов, модульное тестирование и реорганизация кода. Передовые методики и приемы гибкой разработки приспосабливающегося к изменениям кода обсуждаются на конкретных примерах, а в конце книги — на практическом примере отдельного проекта. Книга рассчитана на читателей, имеющих опыт программирования на C# в Visual Studio и на платформе .NET Framework.

ISBN 978-5-8459-1991-5 **в продаже**

Проектирование процесса проектирования: *записки компьютерного эксперта*

Фредерик П. Брукс, мл.

www.williamspublishing.com

Эффективное проектирование лежит в основе любой разработки, начиная от программного обеспечения и заканчивая техническими устройствами и строительными объектами. Но что мы действительно знаем о процессе проектирования? Что приводит к созданию эффективного, изящного проекта? Эти вопросы и рассматриваются в данной книге.

В своих новых эссе Фредерик Брукс точно определяет особенности, присущие всем проектам разработки, а также показывает, какие процессы и подходы с наибольшей вероятностью позволяют добиться превосходства. Автор прослеживает развитие понятия процесса проектирования, рассматривает особенности проектирования при непосредственном общении и дистанционном взаимодействии сотрудников, а также показывает, каковыми на самом являются великие проектировщики. Он исследует тончайшие нюансы процессов проектирования, в том числе бюджетные ограничения многих типов, эстетические требования, условия проведения проектирования и инструментальные средства, подтверждая справедливость своих утверждений на примерах собственных реальных достижений или неудач, от строительства дома до создания операционной системы для IBM/360.

ISBN 978-5-8459-1792-8 **в продаже**

DOMAIN-DRIVEN DESIGN
ПРЕДМЕТНО-ОРИЕНТИРОВАННОЕ ПРОЕКТИРОВАНИЕ
структуризация сложных программных систем

Эрик Эванс

Классическая книга Э. Эванса посвящена наиболее общим вопросам объектно-ориентированной разработки программного обеспечения: структуризации знаний о предметных областях, применению архитектурных шаблонов, построению и анализу моделей, проектированию программных объектов, организации крупномасштабных структур, выработке общего языка и стратегии коммуникации в группе.

Книга предназначена для повышения квалификации программистов, в частности, по методикам экстремального программирования и agile-разработки. Может быть полезна студентам соответствующих специальностей.

www.williamspublishing.com

ISBN 978-5-8459-1597-9 в продаже

ПРИМЕНЕНИЕ UML 2.0 И ШАБЛОНОВ ПРОЕКТИРОВАНИЯ, третье издание

Крэг Ларман

В книге вы найдете новые сведения об итеративном и гибком моделировании, шаблонах проектирования GRASP и GoF, прецедентах, архитектурном анализе и многоуровневой архитектуре, а также рефакторинге, разработке на основе обязанностей и многих других вопросах. Весь материал рассматривается в контексте использования унифицированного процесса (UP) как легкого и гибкого подхода к разработке совместно с приемами из других итеративных методов, таких как XP и Scrum.

Данная книга будет прекрасным руководством для как для новичков, так и для специалистов, кто интересуется вопросами ООА/П, языком моделирования UML 2 и самыми современными эволюционными подходами к разработке программного обеспечения.

www.williamspublishing.com

ISBN 978-5-8459-1185-8 **в продаже**