Содержание

Положительные отзывы к первому изданию

Подробный и обширный справочник со множеством замечательных рекомендаций. Здесь вы узнаете, с чего начать и как правильно применять методы BDD.

– *Фердинандо Сантакроче* (Ferdinando Santacroce),
разработчик на C#, CompuGroup Medical Italia

С помощью этой книги вы изучите BDD от начала до конца и сможете сразу же начать использовать эту методику.

– *Дрор Хелпер* (Dror Helper),
старший консультант, CodeValue

Эта книга для всех, кто желает увидеть, как BDD реализуется на практике. Автор показывает множество полезных методов, инструментов и понятий, которые помогут работать еще продуктивнее с помощью BDD.

– *Карл Метивье* (Karl Métivier),
консультант по методам гибкой разработки, Facilite Informatique

Главное и полное пошаговое руководство по BDD.

– *Марк Блюмнер* (Marc Bluemner),
руководитель отдела контроля качества, Liquidlabs GmbH

Я настоятельно рекомендую эту книгу всем своим коллегам, студентам и инженерам-программистам, заинтересованным в разработке программного обеспечения высочайшего качества.

– *Дэвид Кабреро Соуто* (David Cabrero Souto),
директор Research Group Madsgroup, университет Коруньи

Предисловие

Добро пожаловать во второе издание замечательной книги Джона Фергюсона Смарта (John Ferguson Smart) «BDD в действии». Когда в 2014 году вышло первое издание, я испытал смесь облегчения и восторга от того, что кто-то наконец взял на себя тяжелый труд по описанию методов, инструментов и приемов BDD. Джон тщательно, вдумчиво и подробно задокументировал все, что он видел и пережил как практик, консультант и преподаватель, и я с большим энтузиазмом согласился написать предисловие, которое представило бы эту книгу миру.

С тех пор мир сильно изменился. Глобальная пандемия привела к беспрецедентному распространению слова «беспрецедентный». Команды и организации внедряют распределенные и гибридные модели работы, что делает совместную работу одновременно более важной и более сложной, чем прежде.

Разработка на основе поведения (Behavior-Driven Development, BDD), кажется, идеально подходит для этого нового мира. Ее ориентация на постоянное общение в течение всего цикла разработки продукта позволяет организовать живую, постоянно меняющуюся документацию в виде артефактов, общих для членов команды и других заинтересованных сторон, разделенных географическими и часовыми поясами. Команда может согласовать функцию, обсудить ее область действия в виде названий сценариев («Там, где...») и подробно рассмотреть критерии приемки, при этом автоматически получая гарантию, что предмет договоренности действительно соответствует поведению приложения. Свяжите это с управлением версиями и жизненным циклом, и вы окажетесь на пути к полному и постоянному соблюдению требований!

В этом обновленном издании Джон и Ян пересмотрели все его содержание, улучшили ясность и последовательность объяснений как для тех, кто читает впервые, так и для тех, кто уже читал первое издание. Но мир не стоит на месте; с 2014 года в мире BDD произошло несколько интересных новых событий.

Example Mapping – это простой, но мощный способ изучения особенностей, выявления неопределенностей и фиксации предположений, бизнес-правил, вопросов и, конечно же, примеров. Признаюсь, когда мне впервые об этом рассказали, я в растерянности отреагировал словами: «Простите, что?» Этот метод казался слишком простым, чтобы быть полезным. Но впоследствии он стал одним из моих любимых инструментов BDD.

Модель сценария – еще один метод, который становится очевидным, если громко заявить о нем. Большинство фреймворков автоматизации пользовательского интерфейса используют язык элементов управления – кнопки, поля, формы и т. д., – известный как модель страницы. Сценарий переворачивает эту идею с ног на голову и говорит: «Почему бы не описать взаимодействие с пользовательским интерфейсом на предметном языке бизнеса, как это делается повсюду?» Вам определенно понравится этот метод!

Джон и Ян описывают эти и другие ценные методы с присущей им ясностью и подробностями, приводя действующие примеры, которые помогут читателю не только познакомиться с теорией, но и попрактиковаться. Я поймал себя на том, что согласно киваю головой, читая эту книгу, а также довольно часто произношу: «Ага!»

Я рад, что BDD по-прежнему пользуется большой популярностью, спустя почти 20 лет (!), и я благодарен Джону и Яну за подготовку второго издания такого всеобъемлющего руководства.

– *Дэниел Терхорст-Норт* (Daniel Terhorst-North),
практикующий консультант

Вступление

Начав работу над этим новым изданием «BDD в действии» в 2019 году, я думал, что это будет легко: отразить несколько обновлений в библиотеках и, возможно, добавить пару новых разделов, посвященных более свежим методам обнаружения требований.

Но когда я перечитал все, что написал о BDD в 2013–2014 годах, я понял, что с тех пор *многое* изменилось. Перефразируя персонажа Роя Шайдера (Roy Scheider) в фильме «Челюсти», мне нужна была книга побольше. Основные принципы остались прежними, поскольку фундаментальные идеи, лежащие в основе BDD, столь же актуальны, как и прежде.

Но сам подход к BDD значительно изменился. Мы научились эффективнее проводить сеансы выявления требований с помощью таких методов, как Example Mapping, Feature Mapping и Journey Mapping. Мы также видели, что многие команды неверно понимают суть BDD и страдают от этого, поэтому нам показалось полезным дополнительно разъяснить некоторые основные принципы. В 2015 году я впервые познакомился с моделью сценария, и это стало для меня поворотным моментом, после которого я стал писать более чистый и удобный в сопровождении код автоматизации тестирования.

JavaScript значительно вырос со времени первого издания; я объединился с Яном Молаком (Jan Molak), автором «Serenity/JS» (реализации модели сценария на TypeScript), чтобы глубже изучить техническую сторону практического применения BDD в проектах на JavaScript и TypeScript.

В этом издании ни одна глава не осталась нетронутой, многие были полностью переписаны и появилось несколько совершенно новых. Наслаждайтесь!

Об этой книге

Цель этой книги – помочь командам начать применять эффективные практики BDD. Дать вам полное представление о том, как методы BDD применяются на всех уровнях процесса разработки программного обеспечения, включая выявление и определение требований высокого уровня, реализацию функциональных возможностей приложения и написание выполняемых спецификаций в форме автоматизированных приемочных и модульных тестов.

Кому адресована эта книга

Эта книга рассчитана на широкую аудиторию. Ее могут прочитать и те, кто совершенно незнаком с BDD, и те, кто уже пытается внедрить методику BDD или связанные с ней практики, такие как разработка на основе приемочных испытаний или спецификация на примерах. Эта книга предназначена для всех, кто испытывает сложности из-за несогласованных и постоянно меняющихся требований, вынужден терять время на устранение дефектов, а также желает повысить качество своих продуктов. Она адресована практикам, чья работа состоит в том, чтобы помогать этим людям, и всем, кто разделяет страсть к поиску методов создания и доставки программного обеспечения.

Разные специалисты извлекут пользу из этой книги:

- *бизнес-аналитики и тестировщики* познакомятся с более эффективными способами выявления требований в сотрудничестве с пользователями и передачи этих требований командам разработчиков;
- *разработчики* узнают, как писать более качественный и удобный в сопровождении код с меньшим количеством ошибок, как сосредоточиться на написании кода, приносящего реальную пользу, и как создавать наборы автоматизированных тестов, служащие документацией для всей команды;
- *руководители проектов и заинтересованные стороны от бизнеса* узнают, как помочь командам создавать более ценное программное обеспечение для бизнеса.

Структура книги

Книга разделена на три части, каждая из которых посвящена различным аспектам BDD.

- Часть I представляет мотивацию, происхождение и общую философию BDD и завершается кратким практическим введением

в применение BDD в реальном мире. Эта часть поможет членам проектных команд и заинтересованным сторонам получить четкое представление о том, что такое BDD на самом деле.

- Часть II посвящена сотрудничеству. Здесь вы узнаете, как методы BDD помогают командам эффективно анализировать требования, а также выявлять и описывать возможности, которые принесут реальную пользу организации. Эта часть закладывает концептуальную основу для остальной книги и представляет ряд важных методов анализа требований.
- Часть III содержит подробное техническое описание практик BDD. В ней рассматриваются методы надежной и устойчивой автоматизации приемочных тестов, изучается ряд инструментов BDD для разных языков и платформ и демонстрируется, как BDD помогает разработчикам писать более чистый, структурированный и качественный код. Это сугубо практический раздел. Последняя глава части III немного выбивается из строя и рассматривает более широкую картину BDD в контексте управления проектами, документирования продукта, отчетности и интеграции в процесс сборки.

В большинстве практических примеров, представленных в книге, используются языки и инструменты на основе Java, но мы также рассмотрим примеры инструментов BDD для JavaScript и TypeScript. Однако обсуждаемые нами подходы в целом применимы к любому языку.

Из-за широкой направленности книги различные разделы могут оказаться более или менее применимыми к вашей повседневной работе. Например, бизнес-аналитики могут найти материал по анализу требований более актуальным, чем главы, посвященные практике программирования. В табл. 1 представлено (очень) приблизительное руководство по разделам, которые разные специалисты могут посчитать особенно полезными.

Таблица 1. Примерный состав целевой аудитории для каждого раздела этой книги

	Бизнес-аналитики	Тестировщики	Разработчики	Руководители проектов
Часть I	☑ ☑ ☑	☑ ☑ ☑	☑ ☑ ☑	☑ ☑ ☑
Часть II	☑ ☑ ☑	☑ ☑ ☑	☑ ☑	☑ ☑ ☑
Часть III (главы 11–15)	☑	☑ ☑ ☑	☑ ☑ ☑	☑
Часть III (глава 16)	☑ ☑ ☑	☑ ☑ ☑	☑ ☑	☑ ☑

Предварительные условия

Предварительные условия, необходимые для чтения «BDD в действии», различны для разных частей книги.

- *Части I и II (BDD высокого уровня)* не требуют особых технических знаний; они нацелены на всех членов команд и знакомят

с общими принципами BDD. Базовое понимание практик гибкой разработки будет полезным.

- *Часть III (BDD и автоматизация тестирования)* требует знания программирования. Большинство примеров написаны на Java или JavaScript. Однако все они предназначены лишь для иллюстрации концепций и практик, а не как исчерпывающее описание какой-либо одной технологии. При чтении этой части вам пригодятся практические знания следующих технологий:
  - Maven – в примерах кода на Java/JVM используется Maven, однако вам достаточно будет иметь лишь поверхностные знания (умение создавать проекты Maven);
  - HTML/CSS – разделы по тестированию пользовательского интерфейса с использованием Selenium/WebDriver требуют базового понимания организации HTML-страниц, умения применять селекторы CSS и некоторого знакомства с XPath;
  - веб-сервисы Restful – разделы, посвященные тестированию веб-сервисов, требуют некоторого понимания особенностей реализации веб-сервисов и клиентов веб-сервисов;
  - JavaScript – раздел, посвященный тестированию JavaScript и приложений на JavaScript, требует достаточного хорошего понимания принципов программирования на JavaScript.
- *Глава 16 (живая документация)* – этот раздел носит общий характер и не имеет реальных технических требований.

О примерах кода

Эта книга содержит множество примеров исходного кода, иллюстрирующих применение различных инструментов и методов. Исходный код в листингах или в тексте отображается `моноширинным шрифтом`. Связанные понятия в тексте, например имена классов или переменных, тоже отображаются этим шрифтом.

Поскольку в данной книге обсуждается множество языков, мы постарались сделать все листинги читабельными и понятными даже тем, кто незнаком с используемым языком. Большинство листингов снабжены комментариями, помогающими понять код, а некоторые – нумерованными ссылками, помогающими быстро отыскивать конкретные строки кода, обсуждаемые в тексте.

Исходный код и другие ресурсы

Эта книга содержит множество примеров исходного кода на разных языках. Исходный код всех примеров доступен для загрузки на GitHub по адресу https://github.com/bdd-in-action/second-edition, в отдельных каталогах по главам. Некоторые примеры обсуждаются в нескольких главах – в таких случаях в каталогах соответствующих глав содержится своя копия исходного кода, обсуждаемого в них. На

GitHub также приводятся ссылки на инструменты и библиотеки, использованные в книге, и другие полезные ресурсы.

Кроме того, желающие могут получить фрагменты выполняемого кода из онлайн-версии этой книги по адресу https://livebook.manning.com/book/bdd-in-action-second-edition. Проекты кода не содержат файлов проектов для конкретных IDE, но они организованы так, чтобы их легко было импортировать в IDE.

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге, – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв на нашем сайте www.dmkpress.com, зайдя на страницу книги и оставив комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу dmkpress@gmail.com; при этом укажите название книги в теме письма.

Если вы являетесь экспертом в какой-либо области и заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу http://dmkpress.com/authors/publish_book/ или напишите в издательство по адресу dmkpress@gmail.com.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы обеспечить высокое качество наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг, мы будем очень благодарны, если вы сообщите о ней главному редактору по адресу dmkpress@gmail.com. Сделав это, вы избавите других читателей от недопонимания и поможете нам улучшить последующие издания этой книги.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательства «ДМК Пресс» и Manning Publications очень серьезно относятся к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконной публикацией какой-либо из наших книг, пожалуйста, пришлите нам ссылку на интернет-ресурс, чтобы мы могли применить санкции.

Ссылку на подозрительные материалы можно прислать по адресу электронной почты dmkpress@gmail.com.

Мы высоко ценим любую помощь по защите наших авторов, благодаря которой мы можем предоставлять вам качественные материалы.

Об авторах

Джон Фергюсон Смарт (John Ferguson Smart) – частый участник международных конференций, консультант, автор и преподаватель, хорошо известный в сообществе гибкой разработки своими многочисленными книгами, статьями и презентациями, особенно в таких областях, как BDD, TDD, автоматизация тестирования, мастерство разработки программного обеспечения и командное сотрудничество.

Главная профессиональная цель Джона – помочь организациям и командам повысить эффективность, сочетая эффективное сотрудничество и техническое совершенство. Джон является создателем и ведущим разработчиком инновационной библиотеки автоматизации тестирования Serenity BDD, а также основателем Serenity Dojo (https://www.serenity-dojo.com/program-overview), онлайн-школы, которая помогает тестировщикам любого уровня стать высокоэффективными инженерами по автоматизации тестирования.

Ян Молак (Jan Molak) – преподаватель и консультант, помогающий клиентам по всему миру улучшить сотрудничество и оптимизировать процессы доставки программного обеспечения посредством внедрения BDD, автоматизации тестирования и современных методов разработки программного обеспечения.

Один из разработчиков шаблона сценариев (Screenplay Pattern) и вообще активный разработчик программного обеспечения с открытым исходным кодом. Ян также является автором фреймворка приемочного тестирования Serenity/JS, Jenkins Build Monitor и множества других инструментов в области непрерывной доставки и тестирования.

Часть I

Первые шаги

Добро пожаловать в мир разработки, основанной на поведении (Behavior-Driven Development, BDD)! Первая часть этой книги поможет вам получить общее представление о мире BDD и первое представление о том, как BDD выглядит в полевых условиях.

В главах 1 и 2 вы узнаете о мотивах и происхождении BDD, а также о том, какое место она занимает в ландшафте гибкой разработки (Agile) и других подходах к разработке программного обеспечения. Вы откроете для себя широкую сферу применения методики BDD и узнаете, как она применяется на всех уровнях разработки программного обеспечения: от выявления и описания высокоуровневых требований до подробностей низкоуровневого программирования. Здесь вы узнаете, насколько важно не только правильно создавать программное обеспечение, но и создать правильное программное обеспечение.

Нам, практикам, нравится опираться на примеры из реальной жизни, поэтому в главе 3 мы покажем, как выглядит BDD в реальном проекте: от выявления требований и автоматизации критериев приемки высокого уровня до построения и проверки дизайна проекта, посредством подготовки точной и актуальной технической и функциональной документации.

К концу первой части вы получите хорошее представление о мотивах и общей сфере применения BDD, а также о том, как эта методика применяется на практике на разных уровнях процесса разработки программного обеспечения.

1 Создание программного обеспечения, которое стоит создавать

Эта глава охватывает следующие темы:

- проблемы, которые решает разработка, основанная на поведении;
- общие принципы и истоки BDD;
- действия и результаты, наблюдаемые в проекте BDD;
- плюсы и минусы BDD.

Эта книга посвящена созданию и поставке программного обеспечения, которое хорошо работает и легко поддерживается, а также, что особенно важно, созданию программного обеспечения, представляющего реальную ценность для своих пользователей. Мы стремимся создавать хорошее программное обеспечение, но мы также должны создавать программное обеспечение, которое стоит создавать.

В 2012 году в ВВС США решили отказаться от крупного программного проекта, стоимость которого превысила 1 млрд долларов. Логистическая система Expeditionary Combat Support System (ECSS) была разработана для модернизации и оптимизации управления цепочками поставок с целью сэкономить миллиарды долларов и обеспечить соответствие новым законодательным требованиям. Но после семи лет разработки система так и не «принесла каких-

либо значительных результатов»[1]. По оценкам BBC, для реализации лишь четверти первоначального объема потребуется дополнительно 1,1 млрд долларов, и решение не может быть развернуто до 2020 года, тогда как законом установлен срок развертывания в 2017 году.

Такое часто случается в индустрии программного обеспечения. Согласно ряду исследований, около половины всех проектов не удается реализовать по каким-либо существенным причинам. Ежегодный отчет *CHAOS Report* компании Standish Group за 2011 год показал, что 42 % проектов были реализованы с опозданием, превысили бюджет или не смогли реализовать все запрошенные функции[2], а 21 % проектов были полностью отменены. В ежегодном опросе Скотта Эмблера (Scott Ambler) об успешности ИТ-проектов используется более гибкое определение успеха, но и по его результатам 30–50 % проектов оканчиваются неудачей[3]. Это соответствует миллиардам долларов, потраченных впустую на написание программного обеспечения, которое не будет использоваться или решать бизнес-задачи, которые оно должно решать.

Но должно ли так быть? Разве нельзя писать программное обеспечение так, чтобы мы могли выявлять и сосредоточивать свои усилия на том, что действительно важно? Разве нельзя объективно оценить, реализация каких функций действительно принесет пользу организации, и использовать наиболее экономичные способы их реализации? Разве мы не можем видеть больше того, что просит пользователь, и создавать то, что ему действительно нужно?

Организации находят правильный путь к решению этих проблем. Многие команды успешно сотрудничают, создавая и поставляя ценное, эффективное и надежное программное обеспечение. И они учатся делать это еще быстрее и эффективнее. В этой книге вы увидите, как это сделать. Мы рассмотрим ряд методов и техник, сгруппированных под общим названием *разработка на основе поведения* (Behavior-Driven Development, BDD).

BDD – это подход к организации сотрудничества, следуя которому, команды обсуждают структурированные примеры и контрпримеры бизнес-правил и ожидаемого поведения, чтобы получить глубокое понимание, какие функции действительно принесут пользу пользователям и бизнесу в целом. Очень часто эти примеры выражаются

---

[1] Chris Kanaracus, «Air Force scraps massive ERP project after racking Up $1 billion in costs», CIO, November 14, 2012, https://www.computerworld.com/article/2493041/air-force-scraps-massive-erp-project-after-racking-up--1b-in-costs.html.

[2] Что отражают эти цифры – нашу способность создавать и поставлять программное обеспечение или способность планировать и оценивать, – является предметом споров в сообществе гибкой разработки. См. книгу Джима Хайсмита (Jim Highsmith) «Agile Project Management: Creating Innovative Products» (Addison-Wesley Professional, 2009).

[3] Scott Ambler, «Surveys Exploring the Current State of Information Technology Practices», http://www.ambysoft.com/surveys/.

в формате выполняемого кода, который служит основой для автоматических приемочных тестов, проверяющих поведение программного обеспечения. BDD помогает командам сосредоточить свои усилия на выявлении, понимании и создании ценных функций, важных для бизнеса, а также гарантирует детальную проработку и надежную реализацию этих функций.

Специалисты, практикующие BDD, обсуждают конкретные примеры поведения системы, чтобы лучше понять, какую ценность функции принесут бизнесу. Этот подход поощряет более тесное сотрудничество между бизнес-аналитиками, разработчиками и тестировщиками программного обеспечения, позволяя им выражать требования в форме, более простой для проверки и понятной как разработчикам, так и представителям бизнеса. Инструменты BDD могут помочь превратить эти требования в автоматизированные тесты, чтобы разработчик мог проверять функции и документировать то, что заложено в приложение.

BDD – это не методология разработки программного обеспечения. Это не замена Scrum, XP, Kanban или любой другой методологии разработки, которую вы сейчас используете. Как вы увидите далее, BDD включает, развивает и совершенствует идеи многих из этих методологий. И независимо от выбранной методологии BDD способна облегчить вам вашу жизнь.

1.1 BDD с высоты 50 000 футов

Так что же предлагает BDD? Вот (немного упрощенная) точка зрения. Допустим, компании Криса понадобился новый модуль для программного обеспечения бухгалтерского учета. Процесс добавления новой функции в это программное обеспечение протекает примерно так (рис. 1.1):

1. Крис рассказывает бизнес-аналитику, какая функция ему требуется.
2. Бизнес-аналитик преобразует запросы Криса в набор требований для разработчиков, описывающих, что должно делать программное обеспечение. Эти требования записываются на естественном языке и хранятся в документах Microsoft Word.
3. Разработчик переводит требования в код и модульные тесты, написанные на Java, C# или каком-то другом языке программирования, реализующие и проверяющие новую функцию.
4. Тестировщик переводит требования в тестовые примеры и использует их для проверки соответствия новой функции требованиям.
5. Затем технические писатели переводят код и тестовые примеры обратно в техническую и функциональную документацию на простом естественном языке.

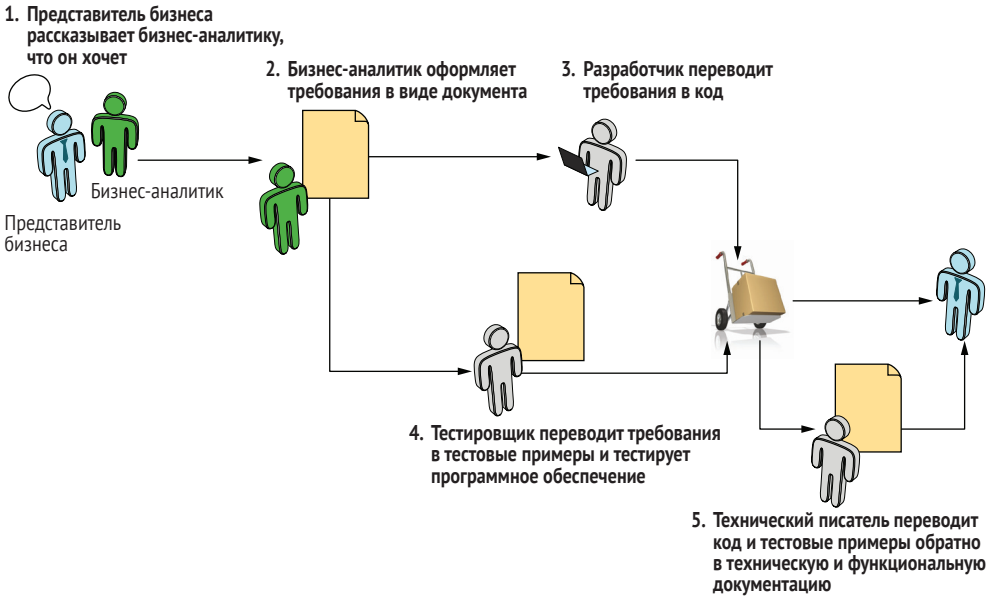

Рис. 1.1 Традиционный процесс разработки открывает массу возможностей для неправильного понимания

На разных этапах этого процесса информация может теряться, неправильно интерпретироваться или просто игнорироваться. Поэтому есть вероятность, что новый модуль будет действовать не так, как требуется, и документация не будет отражать первоначальные требования, полученные аналитиком от Криса.

У Криса есть подруга – Сара. Она управляет другой компанией, которая совсем недавно взяла на вооружение BDD. В команде, практикующей BDD, бизнес-аналитики, разработчики и тестировщики совместно работают над выявлением и определением требований (см. рис. 1.2). Они выражают требования на обобщенном языке, что помогает объединить и сосредоточить усилия команды. Более того, они могут даже превратить эти требования в автоматизированные приемочные тесты, которые не только определяют, как должно вести себя программное обеспечение, но и демонстрируют, что оно действительно ведет себя именно так, как ожидалось. Этот процесс показан на рис. 1.2.

1 Как и Крис, Сара обращается к бизнес-аналитику – Белинде, – чтобы в общих чертах обрисовать новую возможность, которую она хотела бы получить. При этом на встречу приглашаются также разработчик и тестировщик, чтобы те могли из первых уст услышать, что именно нужно пользователям. Дабы снизить риск недоразумений и недопонимания, они вместе составляют примеры желательного и нежелательного поведения новой функции, пытаются сформулировать бизнес-цель и перечисля-

ют, какие особенности и возможности могут помочь в достижении этой цели.

2 Прежде чем начать работу над новой функцией, Белинда снова встречается с разработчиком и тестировщиком. Вместе они обсуждают новую функцию, ключевые бизнес-цели и ожидаемые результаты, а также прорабатывают конкретные примеры и контрпримеры, чтобы глубже понять требования. При обсуждении особенно важных функций на встречу также приглашается Сара.

После этого разговора члены команды записывают ключевые примеры и контрпримеры в удобочитаемом структурированном формате, близком к естественному языку. Эти примеры служат одновременно и спецификациями функций, и основой для автоматических приемочных тестов.

3 Разработчики и тестировщики превращают эти «выполняемые спецификации» в автоматизированные приемочные тесты, которые, в свою очередь, помогают управлять процессом разработки и определять момент завершения работы над функцией.

4 Когда программное обеспечение начинает благополучно проходить все автоматические приемочные тесты, команда получает конкретное доказательство, что функция действует в точ-

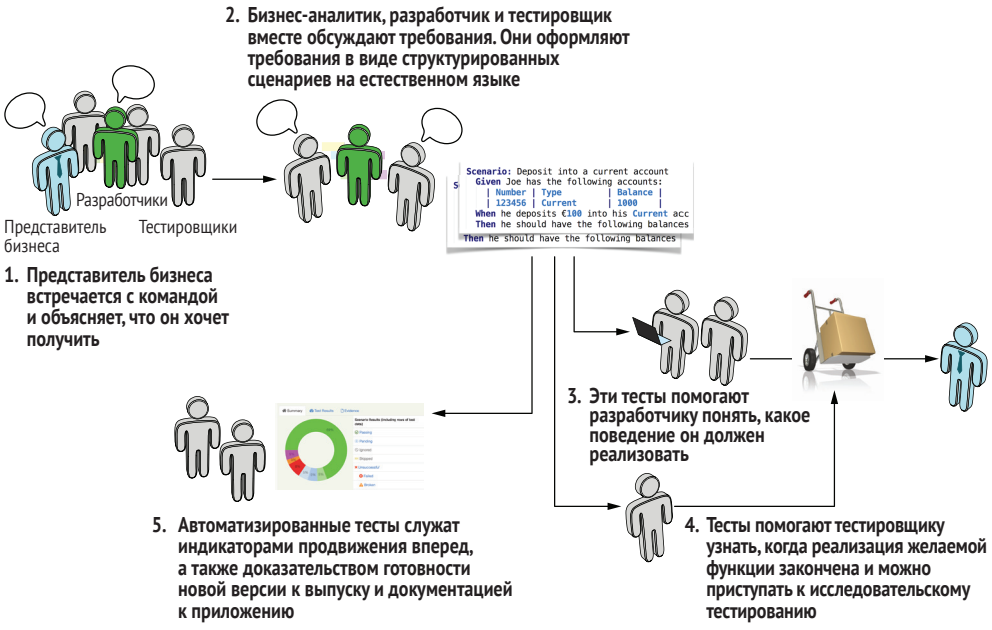

Рис. 1.2 В BDD широко используется обсуждение бизнес-правил и примеров, выраженных в форме, которую легко автоматизировать, благодаря чему уменьшается вероятность потери информации и недопонимания

ном соответствии с требованиями, определенными на этапе 2. Тестировщик может использовать результаты этих тестов как отправную точку для ручного и исследовательского тестирования, которое в любом случае необходимо провести.

5 Автоматизированные тесты также могут служить документацией к продукту, предоставляя точные и актуальные примеры работы системы. Сара может просмотреть отчеты о тестировании, чтобы узнать, какие возможности реализованы и соответствует ли их поведение ее ожиданиям.

В отличие от команды Криса, команда Сары активно общается и составляет примеры, чтобы уменьшить риск потери информации при переводе. Каждый этап после шага 2 основывается на спецификациях, написанных в структурированном виде, понятном для представителей бизнеса, и конкретных примерах, предоставленных Сарой. Благодаря этому почти полностью устраняется всякая двусмысленность при переводе первоначальных требований клиента в код, отчеты и документацию.

Все эти моменты мы подробно обсудим в оставшейся части книги. Вы узнаете, как получить качественный и надежный код, тщательно протестированный и хорошо задокументированный, как писать эффективные модульные тесты и определять значимые критерии приемки и как гарантировать правильное поведение реализуемых вами функций и их реальную полезность для пользователей и бизнеса.

1.2 Какие проблемы вы пытаетесь решить?

Программные проекты терпят неудачу по многим причинам, но наиболее важные делятся на две большие категории:

- неправильное создание программного обеспечения;
- создание неправильного программного обеспечения.

Рисунок 1.3 иллюстрирует их в виде диаграммы. Вертикальная ось представляет, *что* создается, а горизонтальная – *как* создается. Если вы работаете неправильно (горизонтальная ось *как*) и пишете программное обеспечение как попало, то в итоге получите глючный и ненадежный продукт, который к тому же будет трудно изменять и поддерживать. Если вы создаете неправильное программное обеспечение (вертикальная ось *что*), не понимая, какие функции действительно нужны бизнесу, то получите продукт, который никому не нужен.

Рис. 1.3 Для успеха проекта необходимо правильно создавать правильное программное обеспечение

1.2.1 Правильное создание программного обеспечения

Многие проекты терпят неудачу из-за проблем с качеством программного обеспечения. Внутреннее качество программного обеспечения по большей части незаметно для нетехнических пользователей, но последствия некачественного программного обеспечения могут быть весьма болезненными. Как показывает наш опыт, приложения, плохо спроектированные, плохо написанные или не имеющие хороших автоматизированных тестов, обычно содержат ошибки, их трудно поддерживать, трудно изменять и трудно масштабировать.

Мы видели очень много приложений, простое изменение и расширение которых занимало слишком много времени. С каждым разом разработчики тратят все больше и больше времени на исправление ошибок, а не на работу над новыми функциями, что затрудняет быструю реализацию новых возможностей. Новым разработчикам требуется больше времени, чтобы освоиться и приступить к продуктивной работе, просто потому что код труден для понимания. Также становится все сложнее добавлять новые возможности, не нарушая работу существующего кода. Техническая документация (если

таковая имеется) неизбежно устаревает, и команды оказываются неспособными быстро добавлять новые функции, поскольку перед каждым выпуском требуется тратить много времени на ручное тестирование и исправление ошибок.

Организации, использующие высококачественные технические приемы, могут поведать другую историю. Мы видели команды, применяющие такие методы, как разработка через тестирование, чистое кодирование, оперативное документирование и непрерывная интеграция, которые регулярно сообщают о полном или почти полном отсутствии дефектов и выпускают код, простой для сопровождения и модификации при появлении новых требований. Эти команды также имеют возможность сохранять высокий темп добавления новых возможностей, поскольку автоматические тесты гарантируют сохранность работоспособности существующих функций. Они реализуют функции быстрее и точнее, чем другие команды, поскольку им не приходится проводить длительные сеансы исправления ошибок и непредсказуемых побочных эффектов, появляющихся после внесения изменений. А полученное приложение получается более простым в сопровождении.

Обратите внимание, что не существует волшебной формулы создания высококачественного и простого в сопровождении программного обеспечения. Разработка ПО – сложная сфера деятельности, в ней слишком много человеческого фактора, и популярные методы, такие как разработка через тестирование, чистое кодирование и автоматическое тестирование, не гарантируют автоматически получение хорошего результата. Но исследования показывают более сильную корреляцию между бережливыми и гибкими практиками и оценкой успешности проектов[1] по сравнению с традиционными подходами. Другие исследования выявили корреляцию между практиками разработки через тестирование, уменьшением количества ошибок[2] и улучшением качества кода[3]. Безусловно, писать высококачественный код можно, и не практикуя такие методы, как разра-

---

[1] См., например, статью Скотта Уэмблера (Scott Wambler) «2018 IT Project Success Rates Survey Results», http://www.ambysoft.com/surveys/success2018.html.

[2] См., например, статью Начиаппана Нагаппана (Nachiappan Nagappan), Э. Майкла Максимилиана (E. Michael Maximilien), Тирумалеша Бхата (Thirumalesh Bhat) и Лори Уильямс (Laurie Williams) «Realizing quality improvement through test driven development: results and experiences of four industrial teams», https://www.microsoft.com/en-us/research/wp-content/uploads/2009/10/Realizing-Quality-Improvement-Through-Test-Driven-Development-Results-and-Experiences-of-Four-Industrial-Teams-nagappan_tdd.pdf.

[3] Тезисы к диссертации Роба Хилтона (Rod Hilton), «Quantitatively Evaluating Test-Driven Development by Applying Object-Oriented Quality Metrics to Open Source Projects» (PhD thesis, Regis University, 2009), http://www.rodhilton.com/files/tdd_thesis.pdf.

ботка через тестирование и чистое кодирование, однако команды, применяющие их, чаще создают высококачественный код.

Но одного лишь создания высококачественного программного обеспечения недостаточно для успеха проекта. Программное обеспечение также должно приносить пользу пользователям и бизнесу.

1.2.2 Создание правильного программного обеспечения

Программное обеспечение не разрабатывается в вакууме. Программные проекты являются частью более широкой бизнес-стратегии, и чтобы приносить пользу, они должны соответствовать бизнес-целям организации. В конце концов, создаваемое вами программное решение должно помогать пользователям эффективно достигать своих целей. Любые усилия, не способствующие достижению этой цели, пропадают даром.

На практике часто бывает много отходов. Во многих проектах время и деньги тратятся на создание функций, которые никогда не используются или имеют лишь незначительную ценность для бизнеса. Согласно исследованиям CHAOS, проведенным в Standish Group[1], в среднем около 45 % функций, выпущенных в производство, никогда не используются. Даже такие, казалось бы, предсказуемые проекты, как миграция программного обеспечения с устаревших платформ на более современные, имеют свою долю функций, требующих обновления или ставших ненужными. Если вы не до конца понимаете цели, стоящие перед вашим клиентом, то высок риск добавить в приложение хорошо написанные и идеально работающие функции, которые не приносят или почти не приносят пользы конечному пользователю.

С другой стороны, многие проекты в конечном итоге приносят небольшую выгоду бизнесу или вообще не приносят ее. Они не только содержат малополезные функции, но и не предлагают даже минимальных возможностей, которые сделали бы проект жизнеспособным.

Создание правильного программного обеспечения усложняется из-за одного часто упускаемого из виду факта: на ранних этапах проекта обычно неизвестно, какие функции нужны[2].

Как мы увидим далее в этой книге, BDD – очень эффективный способ решения обеих этих проблем. И один из основных способов – устранение одной из основных причин риска и перерасхода средств в проектах: отсутствие достаточно полного понимания того, что необходимо сделать.

---

[1] В отчете CHAOS Report за 2002 год компания Standish Group указала значение 45 %. Позднее мы встречали другие отчеты с результатами внутренних исследований, где эта цифра была близка к 50 %.

[2] См. отчет «Review of the Queensland Health Payroll System», 2012, http://delimiter.com.au/wp-content/uploads/2012/06/KPMG_audit.pdf.

Последствия неправильного создания и создания неправильного ПО

Влияние плохо понятых требований и плохой реализации кода – это не просто теоретическая концепция; напротив, оно часто бывает до боли конкретным. В декабре 2007 года департамент здравоохранения Квинсленда начал работу над новой системой расчета заработной платы для своих 85 000 сотрудников. Первоначальный бюджет проекта составлял около 6 млн долларов, а датой сдачи был назначен август 2008 года.

Когда решение было внедрено в 2010 году, с опозданием примерно на 18 месяцев, случилась катастрофа. Десяткам тысяч государственных служащих зарплата была начислена неправильно, а некоторым вообще не была начислена. С момента запуска системы более 1000 сотрудников отдела заработной платы вынуждены были выполнять около 200 000 ручных операций каждые две недели, чтобы гарантировать выплату заработной платы персоналу.

В 2012 году независимая проверка показала, что всего на проект было потрачено более 416 млн долларов, а на его исправление необходимо дополнительно 837 млн долларов. Эта колоссальная сумма включала 220 млн долларов только на устранение непосредственных ошибок в программном обеспечении, которые мешали системе выполнять свои основные функции по начислению зарплаты сотрудникам.

1.2.3 Ограниченность знаний: борьба с неопределенностью

Обычная ситуация в разработке программного обеспечения, когда вы не знаете каких-то фактов. Изменение требований нередко происходит во многих программных проектах. Знание и понимание рассматриваемой проблемы, а также способов ее наилучшего решения постепенно увеличиваются по мере продвижения проекта.

В разработке программного обеспечения каждый проект уникален. Всегда появляются новые требования бизнеса, которые необходимо удовлетворить, новые технологические проблемы, которые необходимо решить, и новые возможности, которыми можно воспользоваться. По мере продвижения проекта рыночные условия, бизнес-стратегии, технологические ограничения или просто ваше понимание требований будут меняться, и поэтому вам придется изменить свой курс. Каждый проект – это путь открытий, на котором реальным ограничением является не время и не бюджет, а отсутствие знаний о том, что нужно создать и как это сделать. Когда реальность идет не по плану, вы должны адаптироваться к реальности, а не пытаться вписать реальность в ваш план. «Если местность не соответствует карте, доверяй местности» (пословица швейцарской армии).

Пользователи и заинтересованные стороны обычно знают в общих чертах, каких целей они хотят достичь, и если потратить не-

которое время на беседы с ними, то можно раскрыть эти цели. Они смогут сказать, что им нужна онлайн-система продажи билетов или решение для расчета заработной платы, обслуживающее 85 000 сотрудников. И вы уже на ранних этапах сможете получить общее представление о масштабах приложения, которое требуется создать.

Но детали – это совсем другое. Пользователи могут быстро сформулировать конкретные технические решения своих проблем, но обычно они не знают, какое решение подойдет им лучше всего, и могут не знать, какие решения вообще существуют. Коллективное понимание вашей командой наилучшего способа реализации желаемых возможностей, а также оптимального набора функций для достижения основных бизнес-целей будет расти по мере продвижения проекта.

Как показано на рис. 1.4, методы анализа требований, основанные на планах, предполагают, что вы можете очень быстро узнать почти все, что нужно, о требованиях проекта, а также разработать оптимальный дизайн решения на ранних стадиях. К концу этапа анализа требования утверждаются и закрепляются, и после этого остается только написать код.

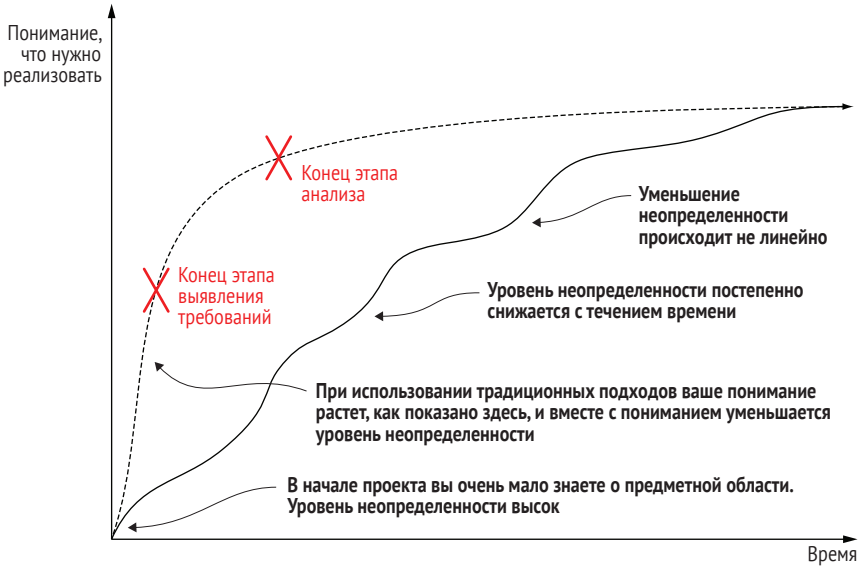

Рис. 1.4 В начале проекта многое неизвестно. Вы постепенно учитесь, восполняя пробелы в знаниях, но путь обучения не является ни линейным, ни предсказуемым

Конечно, реальность не всегда складывается так, как описано. В начале проекта команда разработчиков часто имеет лишь поверхностное представление о предметной области и целях пользователей. Фактически работа команды разработчиков программного обес-

печения должна знать не как создать решение, а как найти лучший способ построить решение.

Коллективное понимание команды естественным образом будет увеличиваться по мере реализации проекта. Со временем вы становитесь менее невежественными. К концу проекта хорошая команда накопит более полные и глубокие знания о потребностях пользователей и сможет активно предлагать функции для реализации, которые лучше подходят для конкретного круга пользователей. Но этот путь обучения не является ни линейным, ни предсказуемым. Мы не можем знать, чего именно мы не знаем, поэтому трудно предсказать, что мы узнаем по ходу проекта.

Для большинства современных проектов разработки программного обеспечения основная задача управления объемом – не в устранении неопределенности путем выявления и фиксации требований как можно раньше, а в управлении этой неопределенностью так, чтобы постепенно отыскивать и предлагать эффективные решения, соответствующие основным бизнес-целям проекта. Как вы увидите далее, одно из важных преимуществ BDD состоит в том, что этот подход предлагает методы, способные помочь справиться с данной неопределенностью и снизить связанный с ней риск.

1.3 Подходит ли BDD для ваших проектов?

BDD хорошо сочетается с другими методологиями гибкой разработки, такими как Scrum, но его также можно использовать с бережливыми подходами, такими как Kanban. Это не отдельная методология, а скорее набор практик, помогающих командам быстрее и эффективнее обнаруживать и изучать потребности бизнеса, а также автоматизировать получение обратной связи об удовлетворении этих потребностей создаваемыми вами функциями. Например, команда Scrum, использующая BDD, будет работать почти так же, как обычная команда Scrum, но при этом для уточнения перечня невыполненных работ они будут применять методы BDD и получать более точную информацию о том, какие функции еще необходимо создать. Также они будут уделять больше внимания автоматизации во время спринта и пытаться писать автоматизированные тесты на основе приемочных критериев для функций, создаваемых во время спринта. Scrum-команда, практикующая BDD, также захочет добавить «автоматизированное приемочное тестирование» к определению пользовательских историй.

BDD хорошо подходит для выявления требований как в новых проектах, так и в тех, которые уже реализуются. Примеры в этой книге посвящены в основном созданию новых функций и новых приложений (за исключением глав 14 и 15, где конкретно рассматривается работа с устаревшими приложениями). Мы поступили так

намеренно, чтобы сделать предметную область более понятной, а примеры – более интересными. Однако подход BDD также можно эффективно применять к существующим приложениям. Оба автора работают в крупных финансовых организациях над сложными проектами. Фактически методы, которые вы изучите в этой книге, применимы к любой области, где требования нетривиальны, где необходимо раскрыть предположения и где сложность и неопределенность лежат под поверхностью каждой истории, которая охватывает почти каждый проект, над которым мы когда-либо работали.

1.4 Что вы узнаете из этой книги

Эта книга даст вам понимание теоретических основ BDD и практические знания о том, как внедрить BDD в вашей собственной организации. Вы узнаете:

- как выяснить реальные требования пользователей, используя методы сотрудничества и правила гибкой разработки, такие как Example Mapping и Feature Mapping;
- как записать эти требования в выполняемом формате (*выполняемые спецификации*), способном действовать как автоматические тесты, и как интегрировать их в процесс сборки;
- как автоматизировать запуск этих выполняемых спецификаций с помощью Java или JavaScript и таких инструментов, как Cucumber;
- как использовать эти выполняемые спецификации для создания постоянно меняющейся, живой документации, которая одновременно проверяет и документирует создаваемые вами функции.

Итоги

- Для успеха проекта разработчики должны создавать надежное и свободное от ошибок программное обеспечение (правильно создавать программное обеспечение), а также создавать функции, несущие реальную пользу бизнесу (создавать правильное программное обеспечение).
- BDD – это подход к совместной разработке, следуя которому, команды используют структурированные обсуждения примеров и контрпримеров, бизнес-правил и ожидаемого поведения, чтобы получить глубокое общее понимание, какие функции принесут пользу пользователям. Очень часто примеры выражаются в выполняемом формате, который служит основой для автоматических приемочных тестов, проверяющих поведение программного обеспечения.

- Специалисты, практикующие BDD, обсуждают конкретные примеры, стараясь понять, какие функции принесут реальную пользу организации.
- Эти примеры составляют основу критериев приемки, которые разработчики используют для определения готовности функции.

В следующей главе мы рассмотрим истоки BDD и более подробно поговорим о ключевых этапах процесса BDD.