Мой жене Дженнифер,
моему пасынку Леону
и нашей дочери Катрин
за то, что они не обижались, когда я
проводил с ними слишком мало времени,
работая над этой книгой

ОГЛАВЛЕНИЕ

ПРЕДИСЛОВИЕ Кента Бека

Существует любопытная симметрия между тем, как в этой книге представлена методология разработки через приемочные тесты (ATDD – Acceptance Test-Driven Development), и тем, как с помощью этой методологии разрабатывается программное обеспечение. Умение выбрать конкретные примеры поведения программы, которые выявляли бы правильное поведение системы в целом, – это особое искусство, как и искусство подобрать такие примеры применения методики, подобной ATDD, которая дала бы читателю возможность освоить ее. Маркус проделал блистательную работу по выбору и представлению примеров.

Читая эту книгу, придется вникать в код. Продвигаясь вперед, вы поймете, как нужно изменить взгляд на проблему, чтобы успешно применять ATDD. В двух словах эта смена мировоззрения заключается в том, чтобы вместо «Вот эту функцию я хочу включить» говорить «А как мы будем ее тестировать? Вот пример». Изучая примеры, вы снова и снова будете встречаться с этим переосмыслением в разных контекстах.

Что мне нравится в построении изложения вокруг примеров кода, так это излучаемая автором уверенность в вашей способности учиться. Это не просто «12 простых правил тестирования веб-приложения», отпечатанных на тонкой папиросной бумаге, которая расползается при первом соприкосновении с влажной реальностью. Вы узнаете о конкретных решениях, принимаемых в конкретных контекстах, о решениях, с которыми вы можете (и обязательно будете, если хотите извлечь из книги максимум пользы) не соглашаться, спорить и делать по-своему.

Ближе к концу книги формулируются общие выводы, подводящие итог принципам, продемонстрированным на примерах. Если вы из тех, кому легче усвоить материал, понимая общие идеи, то начать лучше с этого места. Но в любом случае эффект от прочтения этой

книги прямо будет пропорционален усилиям, которые вы приложите для осмысления примеров.

Одна из слабых сторон первоначального варианта методики разработки через тестирование (TDD) заключается в том, что она может превратиться в технику, которую программист использует для нужд собственно программирования. Некоторые программисты смотрят на TDD более широко, легко переходя в тестах от одного уровня абстрагирования к другому. Но в ATDD такой двусмысленности нет – эта методика предназначена для общения с людьми, которые с языками программирования не знакомы вовсе. Качество отношений с заказчиком – и лежащего в их основе взаимопонимания – способствует повышению эффективности процесса разработки ПО. ATDD может стать шагом в направлении более ясного выражения мыслей, а эта книга – обстоятельное и доступное введение в тему.

– Кент Бек

ПРЕДИСЛОВИЕ
Дэйла Эмери

Не счесть программных проектов, не отвечающих ожиданиям заказчика. На протяжении многих лет я десятки раз слышал, как заказчик объясняет причины провала тем, что *разработчики не обращали внимания на наши пожелания*. А сотни разработчиков, напротив, считают: *«Заказчик не говорит, чего он хочет. По большей части он вообще не знает, чего хочет»*.

Я видел достаточно проектов, чтобы прийти к другому выводу – сформулировать, что должна делать программная система, трудно. Для этого нужно предельно точно говорить и внимательно слушать, а в обычных взаимоотношениях между людьми так бывает редко, да и не очень-то необходимо. Писать хорошие программы трудно. Качественно тестировать программы трудно. Но труднее всего четко объяснить, чего мы хотим от системы.

Методика разработки через приемочные тесты (ATDD) помогает справиться с этой проблемой. С ее помощью вся команда совместно добивается ясности и точного понимания еще до начала разработки. В основе ATDD лежат два приема. Прежде чем приступать к реализации какой-либо функции, члены команды формулируют конкретные примеры ее практического применения. Затем эти примеры переводятся на язык автоматизированных приемочных тестов. Примеры и тесты становятся важной частью точного и разделяемого всеми сторонами описания того, что понимать под словом «сделано» для каждой функции.

Какова цена общего понимания? На семинаре по ATDD один разработчик объяснил это такими словами: «После того как мы начали совместно работать над примерами, мне стало *небезразлично*, что мы делаем. Я наконец-то стал понимать, что и зачем мы создаем. И что еще более важно, я был уверен, что вся команда одинаково понимает, чего мы пытаемся достичь. У нас появилась общая цель – мы стали одной командой».

ATDD не только помогает понять, когда разработку функции следует считать законченной, но и когда приступать к тестированию этой функции (и всех предыдущих). Примеры служат верстовыми столбами на пути к завершению. А поскольку каждый пример описывает ситуацию, значимую для заказчика, то мы можем быть уверены, что не только продвигаемся вперед, но и движемся в нужном направлении.

Ну ладно, я упомянул о нескольких важных особенностях и преимуществах ATDD. Это-то было просто. А вот вопрос посложнее: как всё это организовать в реальном проекте? Ответ оставляю Маркусу Гэртнеру. В этой книге Мартин закатывает рукава и не только рассказывает, но и *показывает*, как ATDD работает на практике. Он позволяет постоять у него за спиной и увидеть, как тестировщики, программисты и бизнес-аналитики думают, применяя принципы и приемы ATDD.

Хочу предупредить об одном подводном камне. В первых главах – когда мы следим, как бизнес-аналитик Билл, тестировщик Тони и программисты Филлис и Алекс описывают и реализуют небольшую программную систему, – материал может показаться чрезмерно упрощенным, даже примитивным. Не поддавайтесь первому впечатлению. В этих главах происходит *много* интересного. Это весьма квалифицированная команда, и некоторые аспекты ее работы на первый взгляд неочевидны. Например, обратите внимание, что при обсуждении требований нет никаких упоминаний о технологии. Члены команды говорят исключительно о бизнес-функциях системы. И заметьте, что когда Тони и Алекс автоматизируют первые тесты, Тони небезуспешно пользуется *отсутствием* у себя опыта программирования. Столкнувшись с непонятной технической деталью, он просит Алекса объяснить ее, а затем они вместе исправляют код, так чтобы он говорил сам за себя. И обратите внимание, как часто Алекс настаивает на сохранении тестов в системе управления версиями, – но только после того как код заработает. Если вы новичок в ATDD, то эти моменты могут показаться неочевидными, но они – неотъемлемая составная часть успеха.

По счастью, чтобы осознать эти тонкие нюансы, нужно всего лишь продолжать чтение. Маркус часто делает паузы, чтобы объяснить, что и почему делает команда. В конце каждой главы он резюмирует, как команда работала совместно, о чем думали ее члены и какие приемы применяли. А в заключительной части книги Маркус сводит все воедино, подробно описывая принципы, на которых зиждется ATDD.

Эта книга – прекрасное введение в методику разработки через приемочные тесты. Заодно она позволяет по-новому взглянуть на ATDD людям, которые уже применяет ее на практике. И наконец, она заслуживает многократного прочтения. Итак, читайте, практикуйтесь и снова читайте. Каждый раз вы будете находить что-то новое и полезное.

– Дэйл Эмери

ВСТУПЛЕНИЕ

Эта книга представляет собой введение в методику, которая получила название «разработка через приемочные тесты», или ATDD (Acceptance Test-Driven Development). Впервые встретив этот термин в 2008 году, я счел его искусственным и излишним. Мне казалось, что это избыточное понятие, потому что тогда я как раз изучал разработку через тестирование и считал, что этого вполне достаточно. В конце концов, зачем бы мне могло понадобиться тестировать программу на соответствие приемочным критериям?

«Каждому воздастся по заслугам»[1] [Wei86]. И вот четыре года спустя я пишу книгу, посвященную именно этой методике. В 2009 году я познакомился с Гойко Аджичем (Gojko Adzic), который только что закончил писать книгу «Bridging the Communication Gap» [Adz09]. Он подарил мне экземпляр, и я тут же, на обратном пути из Лондона, начал читать ее. А когда закончил, имел ясное понимание того, что такое ATDD и почему мы не должны употреблять этот термин.

Но отчего же тогда на обложке книги, которую вы держите в руках, значатся слова «ATDD – разработка через приемочные тесты»?[2]

О названии

Методика ATDD родилась не вчера. Она известна под разными названиями. Вот их неполный перечень:

- разработка через приемочные тесты (Acceptance Test-Driven Development);

1 В оригинале «Time wounds all heels» – парафраз известного изречения «Time heals all wounds» (время лечит все раны). Так называется рассказ американского фантаста Роберта Блоха, написанный в 1942 году и опубликованный в сборнике «Lost in Space and Time with Lefty Feep» 1987 года. Джон Леннон произнес эту фразу в ответ на попытки ФБР депортировать его из США, имея в виду, что время расставит все по своим местам и его притеснителям воздастся по делам их. – *Прим. перев.*

2 Или почему в файле, представляющем электронное издание, встречается именно такая комбинация нулей и единиц?

- разработка на основе поведения (Behavior-Driven Development – BDD);
- спецификирование через пример (Specification by Example);
- гибкое приемочное тестирование (Agile Acceptance Testing);
- тестирование по историям (Story Testing).

На мой взгляд, все эти названия несовершенны. Фраза «разработка через приемочные тесты» подразумевает, что итерацию можно считать законченной, когда все приемочные тесты проходят. Но это не так, потому что ни один набор тестов не дает полного покрытия. Всегда что-то остается непротестированным. Невозможность протестировать программу полностью – хорошо известный феномен. Как говорит Майкл Болтон (Michael Bolton), мы можем быть уверены лишь в одном – если приемочные тесты не проходят, то дело не закончено.

Но я решил привести не аргументы в пользу того или другого названия, а собственно перечень вариантов и дать читателю возможность самому решить, какое название лучше отвечает его представлениям. В конце концов, название не важно, если сама методика решает поставленную задачу. В области разработки программного обеспечения полно неудачных терминов, и такое положение, вероятно, сохранится и в будущем. Программная инженерия, автоматизация тестирования, разработка через тестирование – все эти слова так или иначе вводят в заблуждение. При абстрагировании никогда не следует путать название с предметом. Специалисты знают о недостатках, присущих именованию любого подхода.

Но почему похожие методики называются по-разному? Потому что способы их практического применения могут очень сильно различаться. Имея за плечами опыт консультирования многочисленных коллективов по ATDD, я могу сказать, что общее у них только одно: каждая команда чем-то отличается от всех остальных. Подход, успешно работающий в одной группе внутри одной компании, может привести к полному провалу в другой. Вы никогда не задумывались о смысле слов «зависит от обстоятельств», произносимых консультантом? Он имеет в виду именно это явление.

Работая над книгой «Specification by Example» [Adz11], Гойко Аджич побеседовал более чем с пятьюдесятью коллективами, применяющими ATDD в том или ином виде. Те, что добились успеха, неизменно начинали с базовой методики, а затем пересматривали ее и вносили коррективы с учетом конкретных условий. Начинать с простого процесса и адаптировать его к обстоятельствам, столкнув-

шись с проблемами, – самый гибкий (agile) способ внедрения любой методики. Применяя ATDD, памятуйте о том, что первая попытка вряд ли решит все ваши проблемы. Со временем, накопив опыт, вы сможете видоизменить процесс с учетом конкретных особенностей работы своей команды.

Зачем нужна еще одна книга по ATDD?

Гойко описывает много примеров успешного внедрения ATDD, но я обнаружил существенный пробел во всех имеющихся на сегодняшний день книгах по ATDD. Между опытными командами, давно практикующими тот или иной подход или методику, и командами, только приступающими к ее освоению, имеется огромная разница.

Знакомясь с литературой по ATDD, я наткнулся на несколько книг, в которых ATDD объясняется на продвинутом уровне со ссылками на принципы. Опытному читателю не составит труда применить принципы к конкретному случаю. Но для новичка это совсем не так. Начинающему, чтобы понять, что к чему, нужны более конкретные указания. А уже потом, накопив опыт, он сможет выйти за рамки жестких ограничений, налагаемых методикой.

Новичку проще освоить материал, следуя точному рецепту, но это вовсе не означает, что эта книга – сборник рецептов по ATDD. На рассматриваемых примерах я предлагаю два работоспособных подхода к ATDD и показываю, как думают участники процесса. Начинающий может воспользоваться этими примерами, приступая к внедрению ATDD в своем коллективе. По ходу изложения я буду давать ссылки на более углубленные источники.

Основная идея заимствована из книги Кента Бека «Test-Driven Development: By Example»[3] [Bec02]. Бек приводит два примера разработки через тестирование и в конце объясняет некоторые лежащие в их основе принципы. Книга задумана как вводное описание методики TDD и дает начинающему достаточно материала, чтобы приступить к практическому применению, – в предположении, что TDD можно научиться путем осмысления и практики. В какой-то степени это относится и к данной книге.

3 Кент Бек «Экстремальное программирование: разработка через тестирование». Питер, 2003. *Прим. перев.*

Терминология

В этой книге я пользуюсь терминами из области гибкой разработки ПО. Поскольку не все знают о том, что такое гибкая разработка, я считаю уместным дать краткое описание терминов.

Владелец продукта (product owner). В гибкой методике Scrum определены три роли: команда разработчиков, скрам-мастер (ScrumMaster) и владелец продукта. Владелец продукта отвечает за успех продукта, создаваемого командой. Он устанавливает приоритеты реализации различных функций, обсуждая их со всеми заинтересованными сторонами. Он же играет в команде роль представителя заказчика и в этом качестве принимает решения о деталях – обязательно обговаривая их с другими заинтересованными сторонами.

Итерация, или **забег** (sprint). Основой гибкой разработки является повторяющийся цикл, состоящий из итераций, или – в терминологии Scrum – забегов. Это короткие отрезки, в ходе которых команда реализует одно расширение функциональности, потенциально допускающее включение в готовый продукт. Продолжительность типичной итерации составляет от одной до четырех недель.

Пользовательская история (user story). Это ограниченный набор функций, которые, по мнению команды, можно без напряжения реализовать в ходе одной итерации. Это крохотные срезы функциональности продукта. Обычно команда стремится реализовать за одну итерацию несколько пользовательских историй. За определение историй отвечает представитель заказчика или владелец продукта.

Доска задач (taskboard). В большинстве команд, практикующих гибкую разработку, работа планируется на доске, видной каждому сотруднику. Используются карточки, на которых написано, кто что делает. Доска задач обычно разбита не несколько столбцов – как минимум, «Предстоит сделать», «Делается», «Сделано». На доске отражается текущее состояние работы над продуктом.

Карточка истории (story card). Пользовательские истории обычно записываются на бумажных карточках. По ходу итерации карточки вывешиваются на доску задач.

Ежедневная оперативка (standup meeting, daily Scrum). Не реже раза в день члены команды рассказывают о текущем состоянии дел. Команда собирается на 15 минут и обсуждает, что нужно сделать для завершения оставшихся задач, запланированных на текущую итерацию.

Очередь работ для продукта (product backlog), **очередь работ для забега** (sprint backlog). В методологии Scrum владелец продукта организует еще не реализованные истории в виде очереди работ для продукта. Владелец отвечает за обновление очереди при появлении новых требований. Планируя следующую итерацию, члены команды отбирают задачи для включения в очередь работ для забега. Отобранные из очереди работ для продукта истории автоматически включаются в очередь работ для забега. Чаще всего очередь работ для забега вывешивается на доске задач по завершении планерки.

Рефакторинг. Так называется изменение структуры исходного кода без изменения решаемой им задачи. Я обычно провожу рефакторинг перед внесением изменений. Это позволяет упростить реализацию предстоящих изменений.

Разработка через тестирование (Test-Driven Development – TDD). Методика разработки через тестирование предполагает, что сначала пишется один тест, который не проходит, затем пишется код, необходимый и достаточный для того чтобы тест прошел (и при этом не «сломались» ранее написанные тесты), и после этого код подвергается рефакторингу в преддверии следующего крохотного шажка. TDD – это подход к проектированию, он позволяет улучшить качество кода, потому что по определению создается код, допускающий тестирование.

Непрерывная интеграция, НИ (continuous integration – CI). Это процедура частого включения изменений в исходный код. Сервер сборки собирает всю ветвь, прогоняет все автономные и приемочные тесты, после чего рассылает информацию о новой сборке всем членам команды. В основе НИ лежит какая-то технология автоматизированной сборки, а сама процедура позволяет команде узнать о проблемах в текущей ветви на ранней стадии, а не за час до передачи новой версии заказчику.

Как читать эту книгу

В этой книге описываются как конкретные практические приемы, так и принципы, которые я считаю полезными. Читать книгу можно поразному – всё зависит от уровня вашей подготовки.

Можно читать книгу от корки до корки. Вы узнаете о системе Cucumber, разработке на основе поведения и о том, как тестировать с помощью инструментария ATDD веб-страницы. В нашем первом

примере речь пойдет о команде, где существует четкая граница между программистами и тестировщиками. Вы увидите, что сотрудничество – одно из ключевых условий успеха.

Во второй части мы с вами составим пару. Объединившись в пару, мы сможем компенсировать отсутствие знаний о тестировании или программировании. Мы напишем код приложения, применяя ATDD на практике. Мы познакомимся с основанным на вики каркасом приемочного тестирования FitNesse. Примеры из второй части написаны на Java.

В третьей части приведены рекомендации по внедрению методики. Я даю ссылки на материал для дальнейшего чтения, а также советую, как приступать к делу, и делюсь своими наблюдениями об успешном и не очень успешном опыте применения ATDD в других командах.

В приложениях вы найдете описание как обоих использованных в этой книге инструментов, так и еще одного – достаточно подробные, чтобы приступить к работе. Если раньше вы не встречались с Cucumber или FitNesse, то можете начать с чтения этих приложений.

Опытный читатель может пропустить первые две части и сразу перейти к принципам, изложенным в третьей части. Возможно, впоследствии вы захотите ввести своих коллег в курс дела – для этого могут пригодиться примеры из первой и второй части.

Можно поступить и по-другому – познакомиться с первыми двумя примерами и сразу же попробовать внедрить простой процесс. Зайдя в тупик, можете обратиться к третьей части – хотя я бы не рекомендовал читать книгу в таком порядке.

Если в вашей команде методика ATDD уже внедрена, то, быть может, вам будет интересно более глубоко изучить часть II, где я объясняю, как управлять разработкой предметного кода, отталкиваясь от примеров.

Это лишь некоторые варианты чтения книги, которые пришли мне в голову. Если вы похожи на меня, то, наверное, подумываете о том, чтобы набрать приведенный в примерах код и «поиграться» с ним. Я подготовил репозиторий на сайте github, содержащий код обоих примеров. Это позволит провести приемочное тестирование «вживую». Если вы где-то застрянете, то сможете заодно использовать этот материал как подсказку. Примеры из первой части находятся по адресу http://github.com/mgaertne/airport, а из второй – по адресу http://github.com/mgaertne/trafficlights.

БЛАГОДАРНОСТИ

Такая книга не могла бы состояться без поддержки многих помощников. Прежде всего, я хотел бы поблагодарить Дэйла Эмери, высказавшего очень полезные замечания касательно моего слога. Поскольку английский язык для меня не родной, комментарии Дэйла пришлись весьма кстати.

Отдельное спасибо Кенту Беку. В августе 2010 я говорил с ним по поводу написания книги по ATDD по образцу его книги «TDD by Example». Он же привел меня в издательство Addison-Wesley и представил Кристоферу Гузиковски (Christopher Guzikowski), который оказал мне всемерную поддержку в издании этой книги.

Многие люди принимали участие в рецензировании ранних вариантов текста. Я благодарен Лайзе Криспин (Lisa Crispin), Мэтту Хойсеру (Matt Heusser), Элизабет Хендриксон (Elisabeth Hendrickson), Бретту Шухерту (Brett Schuchert), Гойко Аджичу (Gojko Adzic), Джорджу Динвидди (George Dinwiddie), Кэвину Боди (Kevin Bodie), Олафу Левитцу (Olaf Lewitz), Мануэлю Кюбльбёку (Manuel Küblböck), Андреасу Хавенштайну (Andreas Havenstein), Себастьяну Санитцу (Sebastian Sanitz), Майку Мерчу (Meike Mertsch), Грегору Грамлиху (Gregor Gramlich) и Стефану Кэмперу (Stephan Kämper).

И не в последнюю очередь я хочу поблагодарить свою жену Дженнифер и наших детей Катрин и Леона за поддержку во время работы над этой книгой. Надеюсь в будущем компенсировать то время, в течение которого вы были вынуждены обходиться без мужа и папы.

ОБ АВТОРЕ

Маркус Гэртнер работает тестировщиком, инструктором, тренером и консультантом в компании it-agile GmbH, базирующейся в Гамбурге. Ученик Джерри Вайнберга, Маркус в 2011 году основал германскую рабочую и исследовательскую группу по гибким методикам тестирования. Он также является сооснователем Европейского отделения организации Weekend Testing. Он инструктор и обладатель черного пояса в школе тестирования программного обеспечения Miagi-Do, является участником сообщества пишущих авторов Agile Alliance FTT-Patterns, а также движения Software Craftsmanship. Маркус регулярно выступает на конференциях по гибкой разработке и тестированию по всему миру и активно пишет статьи о тестировании, преимущественно в контексте гибкой разработки. Его личный блог находится по адресу shino.de/blog. Он ведет заказные курсы по ATDD и контекстному тестированию. Он преподавал ATDD тестировщикам без технического образования, а также нескольким программистам.

ЧАСТЬ I.

Парковка в аэропорту