Предисловие

Если зайти на сайт `junit.org`, можно заметить мою цитату: “Никогда в отрасли разработки программного обеспечения столь многие не были так благодарны столь малому объему кода”. Проект JUnit рассматривался как хобби-проект, который любому опытному разработчику под силу написать за выходные. Это так, но подобный взгляд отвлекает от главного. Важность проекта JUnit заключается в фундаментальном смещении фокуса для многих разработчиков: тестирование превратилось в главную и основополагающую часть программирования. Многие предлагали такой подход к разработке и раньше, но именно JUnit сделал основной вклад в этот сдвиг.

Конечно, дело не только в JUnit. Идея была перенесена на большинство языков программирования. В результате целое семейство утилит стало называться xUnit, так как идея давно переросла реализацию для языка Java. (Конечно, все начиналось не с Java: Кент Бек написал этот код для языка Smalltalk значительно раньше.)

Утилиты xUnit, а также их философия, предоставляют огромные возможности командам разработчиков: мощные наборы регрессных тестов позволяют вносить значительные изменения в код с минимальным риском, а при разработке на основе тестов — на ходу пересматривать дизайн продукта.

Но эти возможности сопровождаются новыми проблемами и новыми методиками. Как и любой инструмент, семейство xUnit может использоваться неправильно. Опытные разработчики накопили различные способы использования xUnit для эффективной организации тестов и данных. Как и на заре объектно-ориентированного программирования, большая часть знаний по практическому применению инструментария скрыта в головах опытных пользователей. Без этих “тайных знаний” очень тяжело использовать возможности пакета максимально эффективно.

Около двадцати лет назад была эта проблема была осознана сообществом объектно-ориентированной разработки. Решением проблемы стало описание “тайных знаний” в виде шаблонов. Джерард Месарош был одним из пионеров этого процесса. Когда я начал знакомиться с шаблонами, Джерард был одним из лидеров, у которого мне довелось учиться. Как и многие в мире шаблонов, Джерард стал одним из апологетов технологии экстремального программирования (eXtreme Programming), а значит, работал с утилитами xUnit с самого начала. Логично, что ему пришла в голову идея оформить свои знания эксперта в виде шаблонов.

Мне понравился этот проект с того момента, когда я о нем услышал. (Мне пришлось провести спецоперацию по похищению данной книги у Боба Мартина, так как я хотел включить ее в свою серию.) Как и любая хорошая книга о шаблонах, она содержит информацию, полезную новичкам в данной отрасли. Кроме того, книга

может служить словарем и источником базовой информации для опытных разработчиков, которые пытаются передать свои знания коллегам. Широко известная книга "банды четырех" *Design Patterns* (*Приемы объектно-ориентированного проектирования*, "Питер", 2005 г.) раскрыла секреты объектно-ориентированного проектирования. Предлагаемая вашему вниманию книга имеет такую же ценность с точки зрения применения пакета xUnit.

Мартин Фаулер,
редактор серии,
главный научный сотрудник, ThoughtWorks

Пролог

Ценность самотестирующегося кода

В главе 4 книги *Рефакторинг* [Ref] Мартин Фаулер написал следующее.

> *Если обратить внимание на потраченное разработчиками время, можно заметить, что написание кода на самом деле составляет небольшую его часть. Некоторое время тратится на постановку задачи, некоторое — на проектирование, но большая его часть уходит на отладку. Каждый читатель может вспомнить долгие часы отладки (часто до глубокой ночи). Любой разработчик может рассказать историю об ошибке, исправление которой потребовало целого дня (или даже больше). На самом деле для исправления ошибки много времени не нужно. А вот найти ошибку — совсем другое дело. Не забывайте, что после исправления одной ошибки всегда существует вероятность появления другой, которая остается незаметной очень долго. И еще больше времени потребуется на ее обнаружение.*

Некоторые приложения слишком сложны для тестирования вручную. Часто в таких случаях приходится писать тестовые программы.

В 1996 году автор участвовал в проекте, в котором его задача заключалась в создании инфраструктуры событий, позволяющей клиентскому программному обеспечению регистрироваться в системе и получать уведомление, когда другое приложение генерировало соответствующее событие (шаблон *наблюдатель*, Observer). Я не смог придумать лучшего способа тестирования инфраструктуры, чем создание макета клиентского программного обеспечения. Необходимо было проверить около двадцати различных сценариев, поэтому каждый сценарий был запрограммирован с использованием соответствующего числа наблюдателей, событий и генераторов событий. Поначалу за происходящим приходилось следить, рассматривая вывод на консоль. Очень быстро такой способ контроля оказался слишком утомительным.

Определенная степень лени стимулировала поиск другого решения, связанного с контролем над процессом тестирования. Для каждого теста был создан *словарь* (Dictionary), проиндексированный ожидаемыми событием и получателем и содержащий имя получателя в качестве значения. Как только конкретному получателю отправлялось уведомление о событии, получатель проверял *словарь* на наличие собственной записи с полученным событием. Если такая запись существовала, получатель ее удалял. Если запись не существовала, получатель добавлял запись с сообщением об ошибке, указывающим на уведомление о неожиданном событии.

После запуска всех тестов тестовая программа просто просматривала словарь и выводила непустые записи. В результате запуск всех тестов практически не был связан с накладными расходами. Тесты или успешно завершались без сообщений, или выводили

список сообщений о неудачах. Как потом оказалось, неожиданно для себя автор открыл концепции *подставного объекта* (Mock Object, с. 558) и *инфраструктуры автоматизации тестов* (Test Automation Framework, с. 332).

Первый проект с использованием экстремального программирования

В 1999 году автор был участником конференции OOPSLA, на которой распространялась новая книга Кента Бека *Экстремальное программирование* [XPE]. К этому моменту автор уже применял итеративно-инкрементный подход к разработке и уже понимал ценность автоматизированного модульного тестирования, хотя и не имел опыта его универсального применения. К Кенту Беку автор питал уважение еще со времен первой конференции PLoP (Pattern Languages of Programs) в 1994 году. Совокупность этих факторов убедила автора в желательности применения методологии экстремального программирования в проекте ClearStream Consulting. Почти сразу же после конференции OOPSLA подвернулась возможность применить такой подход в реальном проекте — это было вспомогательное приложение, взаимодействующее с существующей базой данных, но не имеющее собственного пользовательского интерфейса. При этом клиент был готов к применению нетрадиционных методик разработки.

С самого начала применялись все описанные в литературе принципы экстремального программирования, включая программирование в парах, общее владение кодом, а также разработку на основе тестов. Конечно, возникли и сложности при создании тестов для некоторых аспектов поведения приложения, но для большей части кода тесты все же были написаны. После этого в процессе развития проекта стала наблюдаться настораживающая тенденция: реализация одних и тех же решений требовала все больше и больше времени.

После коротких объяснений разработчики стали записывать на каждой карточке задачи время, которое требовалось для создания новых тестов, модификации существующих тестов и создания фактического кода. Очень быстро была выявлена и изолирована интересная тенденция. Как оказалось, время создания новых тестов и написания кода осталось практически неизменным. Но значительно возросло время модификации существующих тестов. Когда разработчик предложил поработать в паре, оказалось, что 90% рабочего времени ушло на модификацию существующих тестов, соответствующих относительно небольшому изменению в функциональности.

Анализ ошибок компиляции и неудачных результатов тестов при добавлении новой функциональности показал, что многие тесты зависели от изменений в методах тестируемой системы. Это никого не удивило. А удивило то, что больше всего проблем возникало на этапе создания тестовой конфигурации и изменения не затрагивали основную логику тестов.

Это стало важным открытием, так как показало, что знания о создании объектов тестируемой системы были равномерно распределены между большинством тестов. Другими словами, тесты слишком много знали о не самых важных аспектах поведения тестируемой системы. Большинство затронутых тестов на самом деле не должны были зависеть от того, как создавались объекты в пределах тестовой конфигурации; их задачей была проверка правильности состояния объектов. При последующей проверке оказалось, что многие тесты создавали одинаковые или почти одинаковые объекты на этапе подготовки.

Очевидным решением проблемы было выделение этой логики в небольшой набор *вспомогательных методов теста* (Test Utility Method, с. 610). На тот момент существовало несколько вариантов.

- Если несколько тестов требовали идентичных объектов, просто использовался метод, который возвращал необходимый объект. Такое решение теперь называется *методом создания* (Creation Method, с. 441).
- В некоторых тестах требовались различные значения атрибутов объекта. В таком случае значение атрибута передавалось в качестве значения параметра для *параметризованного метода создания* (Parameterized Creation Method).
- Некоторым тестам для нормальной работы требовался некорректно сформированный объект, чтобы убедиться в том, что тестируемая система откажется его обрабатывать. Генерация отдельного *параметризованного метода создания* (Parameterized Creation Method) для каждого атрибута излишне раздувала сигнатуру *вспомогательного класса теста* (Test Helper, с. 651), поэтому создавался нормальный объект, значение атрибутов которого модифицировалось с помощью *единственного дефектного атрибута* (One Bad Attribute).

Результатом стал набор тестов и методов, в дальнейшем превратившихся в первые шаблоны автоматизации тестов[1].

Позднее, когда тесты стали завершаться неудачно из-за отказа базы данных принимать объект с уже существующим идентификатором, был добавлен код для генерации уникального ключа. Этот фрагмент был назван *анонимным методом создания* (Anonymous Creation Method, с. 443), что указывало на дополнительную функциональность.

Идентификация проблемы, которая сейчас называется *"хрупким" тестом* (Fragile Test, с. 277), была важным этапом проекта. Последующее определение шаблонов для ее решения спасло проект от возможного неудачного завершения. Без этого открытия пришлось бы как минимум отказаться от уже написанных автоматизированных модульных тестов. В худшем случае тесты снизили бы производительность разработчиков настолько, что клиенты не получили бы необходимый результат. Но в итоге удалось выдать необходимый продукт очень хорошего качества. Да, тестеры (функция тестирования иногда называется контролем качества; такое название является некорректным) находили ошибки в коде, так как некоторые тесты так и не были написаны. Но после того как были написаны отсутствующие тесты, внесение изменений для исправления обнаруженных ошибок практически не требовало усилий.

Тот проект показал, что автоматическое модульное тестирование и разработка на основе тестов действительно работают. С тех пор данные методики используются постоянно.

Хотя разработанные практики и шаблоны применялись в последующих проектах, приходилось сталкиваться с новыми проблемами и задачами. В каждом случае приходилось слой за слоем откапывать корень проблемы и изобретать подходящие решения. Спустя некоторое время развития открытые техники добавлялись в набор подходов к автоматизированному модульному тестированию.

Часть из этих шаблонов была описана в выступлении для конференции XP2001. Обсуждение с коллегами на этой и последующих конференциях показало, что многие

[1] Технически решение не является шаблоном, пока не будет найдено тремя независимыми группами разработчиков.

пользуются такими же или подобными методиками. В результате методы из “практик” превратились в “шаблоны” (повторяющееся решение повторяющейся проблемы). Первое описание **запахов теста** (test smell) было опубликовано на той же конференции. В основе понятия “запах теста” лежало понятие “запах кода”, описанное в книге *Рефакторинг* [Ref][2].

Мотивация

Я глубоко убежден в важности автоматизированного модульного тестирования. Я разрабатывал программное обеспечение без этой технологии около двадцати лет. Инфраструктура xUnit и созданные на ее основе автоматизированные тесты являются одним из заметных шагов в развитии отрасли разработки программного обеспечения. Каждый раз очень неприятно наблюдать, как компания пытается применить автоматизированное модульное тестирование, но терпит неудачу из-за недостатка ключевой информации и необходимых навыков.

Как консультант по разработке ПО в компании ClearStream Consulting я по долгу службы сталкиваюсь с большим количеством проектов. Иногда клиент вызывает консультанта в начале проекта, чтобы “все было правильно”. Но чаще приходится подключаться к проекту, когда он начинает свой полет под откос. В результате пришлось познакомиться с “худшими практиками”, лежащими в основе большинства запахов тестов. Если повезло и меня позвали не слишком поздно, клиент сможет восстановиться после совершенных ошибок. В противном случае клиенту придется выпутываться самому, считая, что разработка на основе тестов и автоматизированное модульное тестирование дают не очень хорошие результаты — и клиент начнет распространять слухи об автоматизированном тестировании как о напрасной трате времени.

Оглядываясь в прошлое, приходится признаться, что все эти проблемы можно было обойти, имея необходимые знания в нужный момент. Но как получить знание, самостоятельно не совершив все ошибки? С точки зрения затраченного на изучение новых технологий времени выгоднее всего пригласить специалиста, который уже обладает необходимыми знаниями. Ппосещение специализированных курсов или чтение книги является менее эффективной (хотя и более дешевой) альтернативой. Надеюсь, что, записав эти ошибки и рекомендованные методы их обхода, я смог помочь многим разработчикам.

Для кого предназначена эта книга

Эта книга написана в основном для разработчиков программного обеспечения (программистов, проектировщиков и архитекторов), которые хотят научиться лучше писать тесты, а также для руководителей и преподавателей, которые хотят понимать, что делают разработчики и зачем им необходимо выделять время на повышение эффективности этой деятельности. Основное внимание уделяется модульным тестам и приемочным тестам, которые автоматизируются с помощью пакета xUnit. Кроме того, ряд шаблонов высокого уровня относится к тестам, автоматизированным на основе отличных от

[2] В книге Мартина Фаулера [Ref] использовались термины “дурной запах” и “неприятный запах”. — *Примеч. ред.*

xUnit технологий. Рик Магридж и Уорд Каннингем написали отличную книгу по инфраструктуре Fit [FitB] и являются сторонниками многих из описанных здесь практик.

Скорее всего, разработчики захотят прочитать книгу от корки до корки, но справочные главы лучше сначала просмотреть, а не вчитываться в каждое слово. Основное внимание нужно уделить общему представлению о существующих шаблонах и принципах их функционирования. К конкретным шаблонам можно будет вернуться, когда в них возникнет необходимость. Обычно описание приводится в нескольких первых подразделах (до раздела "Когда это использовать" включительно) того раздела, который посвященн конкретному шаблону.

Руководителям и преподавателям рекомендуется прочитать часть I, "Общая информация", и, возможно, часть II, "Запахи тестов". Кроме того, желательно прочитать главу 18, "Шаблоны стратегии тестирования", так как в ней описаны решения, смысл которых руководители должны понимать, чтобы поддерживать разработчиков в использовании этих шаблонов. Как минимум руководители должны прочитать главу 3, "Цели автоматизации".

О фотографии на обложке

На обложке каждой книги данной серии размещена фотография моста. Когда Мартин Фаулер предложил включить книгу в свою серию, я задался вопросом: "Какой мост разместить на обложке?" Я подумал о способности тестирования предотвращать катастрофы в работе программного обеспечения и о связи тестирования ПО с мостами. Сразу в голову пришло несколько неудачных мостов, включая "Galloping Gertie" (мост Тэкома-Нэрроуз) и мост "Iron Workers Memorial Bridge" в Ванкувере (мост назван в память о рабочих, погибших при обрушении части конструкции).

После дальнейшего обдумывания мне показалось неоправданным утверждать, что тестирование позволило бы избежать этих ошибок, поэтому мост был выбран, исходя из личных побуждений. На обложке изображен мост "New River Gorge" в Западной Виргинии. Впервые мне довелось перейти его и проплыть под ним в конце 1980-х годов. Архитектура моста также связана с содержимым этой книги: сложная арочная структура под мостом в большинстве случаев скрыта от проезжающих по мосту. Дорога не имеет стыков, и ночью можно проехать мост, даже не заметив его тысячефутовой высоты. Хорошая инфраструктура автоматизации тестов имеет то же свойство: писать тесты легко, так как почти вся сложность скрыта под "дорожным полотном".

Ждем ваших отзывов!

Вы, читатель этой книги, и есть главный ее критик. Мы ценим ваше мнение и хотим знать, что было сделано нами правильно, что можно было сделать лучше и что еще вы хотели бы увидеть изданным нами. Нам интересны любые ваши замечания в наш адрес.

Мы ждем ваших комментариев и надеемся на них. Вы можете прислать нам бумажное или электронное письмо либо просто посетить наш сайт и оставить свои замечания там. Одним словом, любым удобным для вас способом дайте нам знать, нравится ли вам эта книга, а также выскажите свое мнение о том, как сделать наши книги более интересными для вас.

Отправляя письмо или сообщение, не забудьте указать название книги и ее авторов, а также свой обратный адрес. Мы внимательно ознакомимся с вашим мнением и обязательно учтем его при отборе и подготовке к изданию новых книг.

Наши электронные адреса:

E-mail: info@williamspublishing.com
WWW: http://www.williamspublishing.com

Наши почтовые адреса:

в России: 127055, г. Москва, ул. Лесная, д. 43, стр. 1
в Украине: 03150, Киев, а/я 152

Благодарности

Хотя в основном книга создана силами одного автора, многие люди сделали в нее свой вклад. Хотелось бы сразу извиниться перед теми, кто не упомянут здесь.

Те, кто хорошо меня знают, всегда задаются вопросом, где я взял время для написания этой книги. В нерабочее время я обычно занимаюсь различными (кто-то может сказать "экстремальными") видами спорта, например горными лыжами, рафтингом и катанием на горном велосипеде. Лично я не считают эти виды деятельности более экстремальными, чем итеративное и инкрементное программирование. Несмотря на это вопрос о времени, затраченном на написание книги, остается вполне оправданным. Особая благодарность предназначена для моей подруги Хизер Армитидж (Heather Armitage), с которой я занимался большинством перечисленных видов спорта. Она много часов вела машину туда и обратно, пока я сидел с портативным компьютером на заднем сиденье, работая над книгой. Кроме того, хочу выразить признательность Альфу Скрастиньшу (Alf Skrastins), который приглашает своих друзей на лыжные прогулки к западу от Калгари. Отдельная благодарность — администраторам различных лыжных баз, которые разрешали перезаряжать портативный компьютер от их генераторов, что позволяло работать над книгой во время отпуска: Грании Дивайн (Grania Devine) из Selkirk Lodge, Таннис Дакин (Tannis Dakin) из Sorcerer Lodge, а также Дейву Флиру (Dave Flear) и Арону Куперману (Aaron Cooperman) из Sol Mountain Touring. Без их помощи книга создавалась бы намного дольше!

Как обычно, хотелось бы поблагодарить всех рецензентов, официальных и неофициальных. Роберт "Дядя Боб" Мартин (Robert C. "Uncle Bob" Martin) рецензировал ранний черновик. Официальными рецензентами первого "официального" черновика были Лиза Криспин (Lisa Crispin) и Рик Магридж (Rick Mugridge). Джереми Миллер (Jeremy Miller), Алистэр Дьюгид (Alistair Duguid), Майкл Хеджпет (Michael Hedgpeth) и Эндрю Стопфорд (Andrew Stopford) рецензировали второй черновик.

Хотелось бы также поблагодарить моих "пастырей" с различных конференций PLoP, которые высказали свои мнения по поводу описанных здесь шаблонов: Майкла Стала (Michael Stahl), Дэнни Дига (Danny Dig) и особенно Джо Йодера (Joe Yoder). Они предоставили экспертные комментарии по экспериментам с шаблонами. Также отдельная благодарность — группе по языкам шаблонов на конференции PLoP 2004, особенно Юджину Уоллингфорду (Eugene Wallingford), Ральфу Джонсону (Ralph Johnson) и Джозефу Бергину (Joseph Bergin). Брайан Фут (Brian Foote) и группа SAG опубликовали несколько гигабайтов файлов MP3 с рецензиями на ранний черновик книги. Благодаря их комментариям я переписал одну из глав.

За время работы над книгой пришло много сообщений электронной почты с комментариями к материалам, опубликованным на сайте `http://xunitpatterns.com`. Каждый комментарий оказался очень к месту, так как часть опубликованного материала бы-

ла крайне сырой. Среди таких неофициальных рецензентов можно выделить Джавида Джамаи (Javid Jamae), Филипа Нельсона (Philip Nelson), Томаша Гаджевски (Tomasz Gajewski), Джона Херста (John Hurst), Свена Гортса (Sven Gorts), Брэдли Ландиса (Bradley T. Landis), Седрика Беста (Cedric Beust), Джозефа Пелрине (Joseph Pelrine), Себастьяна Бергмана (Sebastian Bergmann), Кевина Резерфорда (Kevin Rutherford), Скотта Амблера (Scott W. Ambler), Джей Би Рейнсбергера (J. B. Rainsberger), Оли Бая (Oli Bye), Дейла Эмери (Dale Emery), Дейвида Нанна (David Nunn), Алекса Чаффи (Alex Chaffee), Буркхардта Хуфнагеля (Burkhardt Hufnagel), Йоханнеса Бродуолла (Johannes Brodwall), Брета Петтикорда (Bret Pettichord), Клинта Шанка (Clint Shank), Сунила Джоглекара (Sunil Joglekar), Рейчел Дейвис (Rachel Davies), Ната Прайса (Nat Pryce), Пола Ходжеттса (Paul Hodgetts), Оуэна Роджерса (Owen Rogers), Амира Кольски (Amir Kolsky), Кевина Лоуренса (Kevin Lawrence), Алистэра Коуберна (Alistair Cockburn), Майкла Фезерса (Michael Feathers) и Джо Шметцера (Joe Schmetzer). Отдельная благодарность выражается Нилу Норвицу (Neal Norwitz), Маркусу Гаэлли (Markus Gaelli), Стефани Дюкасс (Stephane Ducasse) и Стефану Райкхарту (Stefan Reichhart).

Кое-кто отправлял мне электронные письма с описаниями часто применяемого шаблона или специальной функции для используемого пакета xUnit. Большинство из них были вариациями уже документированных шаблонов. В книге они приводятся как псевдонимы или варианты реализации. Было несколько более эзотерических шаблонов, которые не включены в книгу по соображениям объема. За это приношу извинения.

Многие из описанных в этой книге идей взяты из проектов, над которыми мне довелось работать в компании ClearStream Consulting. Нам постоянно создавались предпосылки к поиску более оптимальных способов решения поставленных задач при наличии минимальных ресурсов. Такой подход дал начало многим методикам, описанным в данной книге. Коллегами автора по этому непростому процессу были Дженнита Андреа (Jennitta Andrea), Ральф Боне (Ralph Bohnet), Дейв Браат (Dave Braat), Рассел Брайант (Russel Bryant), Грег Кук (Greg Cook), Джефф Харди (Geoff Hardy), Шон Смит (Shaun Smith) и Томас Таннахилл (Thomas Tannahill). Многие из них стали рецензентами ранних вариантов некоторых глав. Кроме того, Грег является автором большинства примеров кода в главе 25, а Ральф настроил хранилище CVS и автоматизированный процесс компиляции для сайта. Хотелось бы также поблагодарить руководство ClearStream Consulting за разрешение отказаться от консультаций и сконцентрироваться на написании книги и за разрешение на использование упражнений из двухдневного курса “Тестирование для разработчиков” в качестве основы для большинства примеров кода. Итак, благодарности уходят Денису Клилланду (Denis Clelland) и Люку Макфарлану (Luke McFarlane)!

Несколько человек помогали мне работать над книгой, когда все становилось совсем сложно. Они всегда были готовы обсудить по телефону сложный вопрос. Особо хотелось бы отметить Джошуа Кериевски (Joshua Kerievsky) и Мартина Фаулера.

Я выражаю искреннюю признательность Шону Смиту (Shaun Smith) за помощь и за техническую поддержку на начальных этапах работы над книгой. Он обеспечивал работу сайта, создал первые таблицы стилей CSS, научил меня применять Ruby, настроил Wiki-ресурс для обсуждения шаблонов и даже внес свой вклад в их наполнение, пока личные и рабочие факторы не заставили его отойти от писательской составляющей проекта. Каждый раз, когда в описании полученного опыта говорится “мы”, подразумеваемся я и Шон.