Введение

Как-то раз один консультант знакомился с проектом, связанным с разработкой программного обеспечения. Он посмотрел часть готового кода, который был основан на некоторой иерархии классов. Пытаясь пробраться сквозь нее, консультант нашел ее слишком запутанной. Классы в основе иерархии использовали ряд предположений о том, как будут действовать другие классы, и эти предположения были реализованы в коде производных классов. Однако этот код не подходил для всех подклассов, и в результате нередко перекрывался в подклассах. Сократить необходимость в перекрытии кода можно было бы путем внесения некоторых не слишком больших изменений в суперкласс. В некоторых местах выполнялось дублирование поведения суперкласса, связанное с тем, что назначение суперкласса было недостаточно очевидным. Возникали и ситуации, когда ряд подклассов реализовывал одно и то же поведение, которое можно было бы безболезненно переместить выше в иерархии классов.

Рекомендация консультанта заключалась в том, чтобы пересмотреть и подчистить код. Конечно же, это не вызвало радости у руководства. Ведь программа была вполне работоспособной, а сроки поджимали. Поэтому было принято обычное решение — заняться рефакторингом потом. Когда-нибудь. Может быть…

Консультант обратился также к программистам, указав им на найденные недостатки. Программисты отнеслись к замечаниям иначе. По сути, они даже не были виновны — зачастую, чтобы заметить проблему, нужен взгляд со стороны. Потратив пару дней на переделку иерархии, программисты сумели сократить код вдвое без снижения функциональности системы. Результат их вполне удовлетворил, тем более что в итоге облегчилось добавление новых классов в иерархию, а также использование уже имеющихся классов в других местах системы.

Такое поведение программистов вызвало неприятие у руководства. График требовал ускорения работ, а эти два дня, затраченные программистами, не добавили ни одной из функциональностей, которых все еще не хватало в системе. Уже имеющийся код работал — просто он был немного менее понятным и “чистым”. Руководство можно понять. Коммерческий результат приносит работающий код, а не код, который нравится “академической крысе”. Консультант же предложил модифицировать и другие части кода, что могло приостановить работы на пару недель, и лишь ради более красивого кода, а не для расширения его возможностей.

Что вы скажете об этой истории? Кто, по-вашему, был прав? Может, действительно права старая программистская мудрость “Не трогай работающую систему”?

Думаю, вы сразу догадались, что этим консультантом был я. Шестью месяцами позже проект бесславно закрылся, рухнув под собственной тяжестью, — код стал слишком громоздким для отладки или получения приемлемой производительности.

Чтобы возобновить проект, был привлечен другой консультант — Кент Бек, и практически все пришлось писать заново. Кент спроектировал многое совершенно иначе, чем ранее, при этом настояв на постоянном совершенствовании кода с применением рефакторинга. Успех описанного проекта и роль рефакторинга в этом успехе подвигли меня на написание данной книги. Книга позволила мне поделиться опытом и знаниями, приобретенными Кентом и другими специалистами при использовании рефакторинга для повышения качества программного обеспечения.

Что такое рефакторинг

Рефакторинг — это процесс изменения программного проекта, в ходе которого внешнее поведение кода остается неизменным при усовершенствовании его внутренней структуры. Это систематизированный способ очистки кода, минимизирующий возможность появления новых ошибок. По сути, рефакторинг кода представляет собой улучшение проекта уже после того, как этот код написан.

"Улучшение проекта после написания кода" звучит непривычно. При нынешнем понимании процесса разработки программного обеспечения мы сначала создаем проект, а потом пишем код. Первым шагом идет проектирование, а уже затем выполняется кодирование. Со временем код будет изменяться, а целостность системы и ее соответствие первоначальному проекту будет постепенно размываться. Кодирование понемногу перестает быть инженерным искусством и превращается в хакерство.

Рефакторинг же представляет собой нечто совершенно иное. Он позволяет, взяв плохой и беспорядочный проект, превратить его в ясно спроектированный код. Каждый шаг этого преобразования очень прост. Это может быть перемещение поля из одного класса в другой, выделение части исходного текста из метода и ее перемещение в отдельный метод, перемещение некоторых фрагментов кода в том или ином направлении иерархии классов. Кумулятивный эффект таких малозаметных изменений может привести к существенному улучшению программы. Этот процесс оказывается прямой противоположностью описанной выше тенденции постепенной деградации программного проекта.

При рефакторинге требуется баланс между разными этапами работ над изменяемым проектом. Проектирование становится не отдельным начальным этапом разработки, а непрерывным процессом. В ходе работы над проектом вы

постоянно рассматриваете возможность его улучшения. В результате получается программный проект, качество которого не снижается в процессе разработки.

О чем эта книга

Эта книга представляет собой руководство по рефакторингу; она написана для программистов-профессионалов. Моя цель при написании книги — показать, как выполнять рефакторинг управляемо и эффективно. Вы научитесь выполнять рефакторинг так, чтобы не вносить при этом в код новые ошибки, а постоянно улучшать его структуру.

Обычно книги начинаются с введения. Хотя я согласен с этой традицией, мне кажется слишком сложным начинать знакомство с рефакторингом с общего обсуждения или определений. Поэтому я начну с примера. В главе 1, "Первый пример рефакторинга", приводится небольшая программа, в которой имеются обычные недостатки и которая с помощью рефакторинга превращается в более приемлемую объектно-ориентированную программу. Попутно мы рассмотрим как процесс рефакторинга в целом, так и применение некоторых полезных рефакторингов. Эта глава — ключевая для понимания того, чем в действительности является рефакторинг.

В главе 2, "Принципы рефакторинга", я рассказываю об общих принципах рефакторинга более детально; там же я даю некоторые определения и описываю некоторые причины проведения рефакторинга. Здесь же рассматриваются и некоторые проблемы, связанные с рефакторингом. В главе 3, "Запах в коде", посвященной тому, как обнаружить запахи в коде и избавиться от него, мне помогает Кент Бек. Очень большую роль при выполнении рефакторинга играет тестирование, поэтому глава 4, "Создание тестов", посвящена созданию тестов с использованием простого каркаса тестирования Java с открытым исходным кодом.

Основной материал книги, который представляет собой каталог методов рефакторинга, занимает главы с 5 по 12. Этот каталог ни в ком случае нельзя считать исчерпывающим; он представляет собой лишь начало такового каталога. В него включены те рефакторинги, с которыми я сталкивался во время работы в этой области. Когда я хочу выполнить какой-то из рефакторингов, например "Замена условной инструкции полиморфизмом" (с. 271), я обращаюсь к каталогу, чтобы посмотреть, как это сделать наиболее безопасно, шаг за шагом. Надеюсь, вы будете часто возвращаться к этой части книги.

В этой книге описаны не только мои результаты, но и многих других исследователей. Более того, последние главы книги написаны некоторыми из них. Глава 13, "Рефакторинг, повторное использование и реальность", принадлежит перу Билла Опдайка, который пишет о сложностях рефакторинга в коммерческих проектах.

Глава 14, "Инструментарий для выполнения рефакторинга", написанная Доном Робертсом и Джоном Брантом, посвящена автоматизированному инструментарию — будущему рефакторинга. Завершает книгу глава 15, "Заключение", автором которой является выдающийся мастер рефакторинга Кент Бек.

Рефакторинг и Java

В этой книге я везде использую примеры на языке программирования Java. Конечно, рефакторинг может выполняться и для других языков, и, я надеюсь, эта книга будет полезна и тем, кто работает на других языках программирования. Однако я предпочел Java, потому что этот язык я знаю лучше других. Я добавил несколько примечаний о рефакторинге на других языках и надеюсь, что на основе моей книги другие авторы напишут свои книги для других языков программирования.

Для лучшего изложения идей я не вдавался в особо сложные области языка Java. Поэтому в книге нет использования внутренних классов, рефлексии, потоков и многих других мощных возможностей Java. Я хотел как можно доступнее изложить базовые методы рефакторинга, а не блеснуть знаниями языка программирования.

Должен отметить, что приведенные методы рефакторинга не учитывают возможности параллельного или распределенного программирования. В этих областях имеются свои сложности, которые выходят за рамки данной книги.

На кого рассчитана эта книга

Данная книга предназначена для профессиональных программистов. В примерах и обсуждении имеется множество кода, который нужно прочесть и понять. Все примеры написаны на языке программирования Java. Этот язык выбран в силу роста его распространенности и простоты для понимания теми, кто знаком с языком программирования C. Это объектно-ориентированный язык, а объектно-ориентированные механизмы очень способствуют выполнению рефакторинга.

Хотя рефакторинг ориентирован на исходные тексты, он существенно влияет и на проектирование. Руководителям-проектировщикам и архитекторам жизненно необходимо понимать принципы рефакторинга и использовать их в своих проектах. Пожалуй, будет лучше, если рефакторингом будет руководить опытный и уважаемый человек. Он сможет лучше понять принципы, на которых основан рефакторинг, и адаптировать их для конкретного проекта. Это особенно важно в случае языков программирования, отличных от Java, так как приведенные в книге примеры требуют определенной адаптации.

Можно получить пользу от книги, даже не читая ее полностью.

- **Если вы хотите понять, что такое рефакторинг**, достаточно прочесть главу 1, "Первый пример рефакторинга"; приведенный пример должен облегчить ваше понимание.
- **Если вы хотите понять, для чего нужен рефакторинг**, прочтите главы 1 и 2. В них рассказывается о том, что такое рефакторинг и почему он необходим.
- **Если вы хотите узнать, когда следует применять рефакторинг**, прочтите главу 3, "Запах в коде". В ней описаны признаки, говорящие о необходимости применения рефакторинга.
- **Если вы хотите выполнить рефакторинг**, прочтите главы 1–4 полностью. Затем бегло ознакомьтесь с каталогом в главе 5, "На пути к каталогу рефакторингов", чтобы примерно представлять себе его содержание. Вам не нужно вдаваться во все детали. Когда вам понадобится какой-либо из методов рефакторинга, вы сможете прочесть детальную информацию о нем и использовать ее в своей работе. Каталог представляет собой справочный раздел, так что читать все подряд необязательно. Вашего внимания достойны также главы, написанные приглашенными соавторами, в особенности глава 15, "Заключение".

На плечах других

Хочу сразу же сообщить, что эта книга обязана своим появлением тем, чья работа в последнее десятилетие была посвящена развитию рефакторинга. В идеале эту книгу должен был написать кто-то из них, но время и энергия для этой работы нашлись у меня.

Двумя из ведущих сторонников рефакторинга являются Уорд Каннингем (Ward Cunningham) и Кент Бек. Для них рефакторинг давно стал центральной частью процесса разработки, принятой для ежедневного применения с целью получения всех его преимуществ. Именно сотрудничество с Кентом показало мне всю важность рефакторинга и подвигло меня на написание этой книги.

Ральф Джонсон (Ralph Johnson) возглавляет группу в Университете штата Иллинойс в Урбана-Шампань, известную своим вкладом в объектную технологию. Ральф давно является сторонником рефакторинга, как и множество его студентов. Автором первой работы, посвященной рефакторингу, стал Билл Опдайк со своей докторской диссертацией. Джон Брант и Дон Робертс не ограничились словами и создали инструментарий — Refactoring Browser, — предназначенный для выполнения рефакторинга исходных текстов на языке Smalltalk.

Благодарности

Несмотря на использование результатов упомянутых выше исследований, для написания книги мне понадобилась большая помощь. В первую очередь, она была оказана мне Кентом Беком. Первые семена книги были посеяны в баре в Детройте, где Кент рассказал мне о статье, которую он написал для *Smalltalk Report* [5]. Я не только взял из нее многие идеи для главы 1, "Первый пример рефакторинга", но и под ее влиянием начал собирать материал по рефакторингу. Кент помог мне концепцией запаха кода (code smells), а еще поддерживал меня в трудные минуты и делал все от него зависящее, чтобы эта книга увидела свет. Думаю, что сам он смог бы написать эту книгу лучше меня, но, как я уже говорил, время для этой работы нашлось у меня, и мне остается только надеяться, что мой труд не напрасен.

После написания своей части книги мне захотелось поделиться с вами знаниями непосредственно от специалистов в области рефакторинга, и я очень благодарен многим из них за то, что они не пожалели времени и написали для книги свою часть материала. Кент Бек, Джон Брант, Уильям Опдайк и Дон Робертс написали некоторые из глав (самостоятельно или в соавторстве). Кроме того, Рич Гарзанити (Rich Garzaniti) и Рон Джеффрис (Ron Jeffries) добавили полезные врезки.

Любой автор согласится с тем, что для технических книг, таких как эта, очень полезно участие рецензентов. Как обычно, Картер Шанклин (Carter Shanklin) и его команда из Addison-Wesley собрали группу квалифицированных рецензентов, в которую вошли следующие специалисты.

- Кен Ауэр (Ken Auer) из Rolemodel Software, Inc.
- Джошуа Блох (Joshua Bloch) из Sun Microsystems, Java Software
- Джон Брант из Иллинойского университета в Урбана-Шампань
- Скотт Корли (Scott Corley) из High Voltage Software, Inc.
- Уорд Каннингем (Ward Cunningham) из Cunningham & Cunningham, Inc.
- Стефен Дьюкасс (Stephane Ducasse)
- Эрих Гамма (Erich Gamma) из Object Technology International, Inc.
- Рон Джеффрис (Ron Jeffries)
- Ральф Джонсон (Ralph Johnson) из Иллинойского университета
- Джошуа Кериевски (Joshua Kerievsky) из Industrial Logic, Inc.
- Дуг Ли (Doug Lea) из SUNY Oswego
- Сандер Тишлар (Sander Tichelaar)

Все они внесли большой вклад в стиль и точность книги и выявили по крайней мере бо́льшую часть ошибок, которые всегда есть в любой рукописи. Хочу упомянуть два наиболее важных предложения, повлиявших на вид книги. Уорд и Рон заставили меня изменить стиль изложения главы 1, "Первый пример рефакторинга", а Джошуа Кериевски (Joshua Kerievsky) внес предложение дать в каталоге наброски кода.

Кроме официальных рецензентов, книгу читали и комментировали другие люди, сделавшие немало ценных замечаний. В их числе Лейф Беннет (Leif Bennett), Майкл Фезерс (Michael Feathers), Майкл Финни (Michael Finney), Нейл Галарнье (Neil Galarneau), Хишем Газули (Hisham Ghazouli), Тони Гоулд (Tony Gould), Джон Айзнер (John Isner), Брайен Марик (Brian Marick), Ральф Рейссинг (Ralf Reissing), Джон Солт (John Salt), Марк Свонсон (Mark Swanson), Дейв Томас (Dave Thomas) и Дон Уэллс (Don Wells). Есть и другие, которых я здесь не упомянул; приношу им свои извинения и благодарности.

Особенно забавными рецензентами оказались члены печально известной группы читателей из Университета штата Иллинойс в Урбана-Шампань. Поскольку эта книга в значительной мере отражает их работу, я особенно благодарен им за их труд, переданный мне в аудиоформате. Членами этой группы являются Фредрико "Фред" Балагер (Fredrico "Fred" Balaguer), Джон Брант, Ян Чай (Ian Chai), Брайен Фут (Brian Foote), Алехандра Гарридо (Alejandra Garrido), Жийанг "Джон" Хан (Zhijiang "John" Han), Питер Хэтч (Peter Hatch), Ральф Джонсон (Ralph Johnson), Сонгай "Раймонд" Лу (Songyu "Raymond" Lu), Драгос-Антон Манолеску (Dragos-Anton Manolescu), Хироки Накамура (Hiroaki Nakamura), Джеймс Овертерф (James Overturf), Дон Робертс, Чико Шираи (Chieko Shirai), Лес Тайрел (Les Tyrell) и Джо Йодер (Joe Yoder).

Любая хорошая идея требует проверки в серьезной промышленной системе. Я видел, какой эффект оказывал рефакторинг на систему Chrysler Comprehensive Compensation (C3). Хочу поблагодарить всех членов этой команды: Энн Андерсон (Ann Anderson), Эда Андери (Ed Anderi), Ральфа Битти (Ralph Beattie), Кента Бека, Дэвида Брайанта (David Bryant), Боба Коу (Bob Coe), Мари Д'Армен (Marie DeArment), Маргарет Фрончак (Margaret Fronczak), Рича Гарзанити, Денниса Гора (Dennis Gore), Брайена Хакера (Brian Hacker), Чета Хендриксона (Chet Hendrickson), Рона Джеффриса (Ron Jeffries), Дуга Джоппи (Doug Joppie), Дэвида Кима (David Kim), Пола Ковальски (Paul Kowalsky), Дебби Мюллер (Debbie Mueller), Тома Мураски (Tom Murasky), Ричарда Наттера (Richard Nutter), Адриана Панти (Adrian Pantea), Мэтта Сайджена (Matt Saigeon), Дона Томаса (Don Thomas) и Дона Уэллса (Don Wells). Работа с ними утвердила мое понимание принципов и преимуществ рефакторинга. Наблюдения за их работой показали мне, чего позволяет добиться рефакторинг, применяемый в большом проекте на протяжении ряда лет.

Мне также помогали Дж. Картер Шанклин (J. Carter Shanklin) из Addison-Wesley и его команда: Крыся Бебик (Krysia Bebick), Сьюзен Кестон (Susan Cestone), Чак Даттон (Chuck Dutton), Кристин Эриксон (Kristin Erickson), Джон Фуллер (John Fuller), Кристофер Гузиковски (Christopher Guzikowski), Симон Пэймент (Simone Payment) и Женевьев Раевски (Genevieve Rajewski). Работать с хорошим издателем — одно удовольствие! Они оказали мне большую поддержку и помощь.

Говоря о поддержке, хочу заметить, что больше всего неприятностей книга приносит тем, кто находится рядом с ее автором. Я имею в виду мою жену Синди. Спасибо за любовь ко мне, проявлявшуюся даже тогда, когда я прятался в своем кабинете. Но даже там на протяжении всего времени работы над книгой я помнил о тебе!

Мартин Фаулер
Мерлоуз, Массачусетс
fowler@acm.org
http://www.martinfowler.com
http://www.refactoring.com