

**ЭФФЕКТИВНАЯ
РАБОТА С
УНАСЛЕДОВАННЫМ
КОДОМ**

WORKING EFFECTIVELY WITH LEGACY CODE

Michael C. Feathers



Prentice Hall Professional Technical Reference
Upper Saddle River, NJ 07458
www.phptr.com

ЭФФЕКТИВНАЯ РАБОТА С УНАСЛЕДОВАННЫМ КОДОМ

Майкл К. Физерс



Издательский дом “ВИЛЬЯМС”
Москва • Санкт-Петербург • Киев
2009

ББК 32.973.26-018.2.75

Ф50

УДК 681.3.07

Издательский дом “Вильямс”

Зав. редакцией *С.Н. Тригуб*

Перевод с английского и редакция *И.В. Берштейна*

По общим вопросам обращайтесь в Издательский дом “Вильямс” по адресу:
info@williamspublishing.com, http://www.williamspublishing.com

Физерс, Майкл.

Ф50 Эффективная работа с унаследованным кодом. : Пер. с англ. — М. : ООО
“И.Д. Вильямс”, 2009. — 400 с. : ил. — Парал. тит. англ.

ISBN 978-5-8459-1530-6 (рус.)

ББК 32.973.26-018.2.75

Все названия программных продуктов являются зарегистрированными торговыми марками соответствующих фирм.

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

Authorized translation from the English language edition published by Prentice Hall, Inc., Copyright © 2005 Pearson Education, Inc.

All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner and the publisher.

Russian language edition is published by Williams Publishing House according to the Agreement with R&I Enterprises International, Copyright © 2009.

Научно-популярное издание

Майкл Физерс

Эффективная работа с унаследованным кодом

Литературный редактор *Л.Н. Важенина*

Верстка *М.А. Удалов*

Художественный редактор *С.А. Чернокозинский*

Корректор *Л.А. Гордиенко*

Подписано в печать 29.01.2009. Формат 70x100/16.

Гарнитура Times. Печать офсетная.

Усл. печ. л. 25,0. Уч.-изд. л. 21,8.

Тираж 1000 экз. Заказ № 0000.

Отпечатано по технологии СтР
в ОАО “Печатный двор” им. А. М. Горького
197110, Санкт-Петербург, Чкаловский пр., 15.

ООО “И. Д. Вильямс”, 127055, г. Москва, ул. Лесная, д. 43, стр. 1

ISBN 978-5-8459-1530-6 (рус.)

ISBN 0-13-117705-2 (англ.)

© Издательский дом “Вильямс”, 2009

© Pearson Education, Inc., 2005

Оглавление

Введение	21
Часть I. Внутренний механизм изменений в коде	23
Глава 1. Изменения в программном коде	25
Глава 2. Работа с ответной реакцией	31
Глава 3. Распознавание и разделение	41
Глава 4. Модель шва	49
Глава 5. Инструментальные средства	65
Часть II. Изменение программного обеспечения	75
Глава 6. Изменения необходимы, а времени на это нет	77
Глава 7. Изменения до бесконечности	95
Глава 8. Как ввести новое свойство	103
Глава 9. Класс нельзя ввести в средства тестирования	119
Глава 10. Метод нельзя выполнить в средствах тестирования	149
Глава 11. Требуется изменения в коде, но неизвестно, какие методы следует тестировать	163
Глава 12. На одном участке требуется внести много изменений, но следует ли разрывать зависимости со всеми классами, имеющими к этому отношение	181
Глава 13. В код требуется внести изменения, но неизвестно, какие тесты писать	191
Глава 14. Убийственная зависимость от библиотек	201
Глава 15. Приложение состоит из сплошных вызовов интерфейса API	203
Глава 16. Код недостаточно понятен для его изменения	211
Глава 17. У приложения отсутствует структура	215
Глава 18. Когда тестовый код мешает	223
Глава 19. Как благополучно изменить процедурный код	227
Глава 20. Класс слишком крупный и его дальнейшее укрупнение нежелательно	241
Глава 21. Изменение одного и того же кода повсеместно	261
Глава 22. Необходимо изменить гигантский метод, но нельзя написать для него тест	279
Глава 23. Как узнать, нарушают ли что-нибудь изменения в коде	297
Глава 24. Сдаемся — дальнейшее улучшение невозможно	305
Часть III. Методы разрыва зависимостей	307
Глава 25. Способы разрыва зависимостей	309
Приложение. Реорганизация кода	383
Словарь специальных терминов	387
Предметный указатель	389

Содержание

Предисловие	13
Вступление	15
Благодарности	19
Введение	21
Об организации книги	21
От издательства	22
Часть I. Внутренний механизм изменений в коде	23
Глава 1. Изменения в программном коде	25
Четыре причины изменений в программном коде	25
Ввод свойств и исправление программных ошибок	25
Улучшение структуры кода	27
Оптимизация	27
Собирая все вместе	28
Глава 2. Работа с ответной реакцией	31
Что означает блочное тестирование	33
Тестирование на более высоком уровне	36
Тестовое покрытие	36
Алгоритм изменения унаследованного кода	39
Определение точек изменения	39
Нахождение тестовых точек	39
Разрывание зависимостей	39
Написание тестов	40
Внесение изменений и реорганизация кода	40
Остальная часть книги	40
Глава 3. Распознавание и разделение	41
Имитация взаимодействующих объектов	42
Фиктивные объекты	42
Две стороны фиктивных объектов	45
Реализация фиктивных объектов	46
Имитирующие объекты	46
Глава 4. Модель шва	49
Программа как большой лист печатного текста	49
Швы	50
Типы швов	53
Швы предварительной обработки	53
Компоновочные швы	56
Объектные швы	59

Глава 5. Инструментальные средства	65
Инструментальные средства автоматической реорганизации кода	65
Имитирующие объекты	67
Средства блочного тестирования	68
Средство тестирования JUnit	69
Средство тестирования CppUnitLite	70
Средство тестирования NUnit	72
Другие средства блочного тестирования типа xUnit	72
Средства общего тестирования	72
Среда интегрированного тестирования FIT	73
Среда тестирования Fitnesse	73
Часть II. Изменение программного обеспечения	75
Глава 6. Изменения необходимы, а времени на это нет	77
Почкование метода	79
Преимущества и недостатки	82
Почкование класса	82
Преимущества и недостатки	86
Охват метода	86
Преимущества и недостатки	89
Охват класса	89
Резюме	94
Глава 7. Изменения до бесконечности	95
Понимание кода	95
Время задержки	96
Разрыв зависимостей	97
Зависимости компоновки	97
Резюме	101
Глава 8. Как ввести новое свойство	103
Разработка посредством тестирования	103
Написание контрольного примера непрохождения теста	104
Подготовка к компиляции	104
Подготовка к прохождению	104
Исключение дублирования кода	105
Написание контрольного примера непрохождения теста	105
Подготовка к компиляции	105
Подготовка к прохождению	106
Исключение дублирования кода	106
Написание контрольного примера непрохождения теста	106
Подготовка к компиляции	107
Подготовка к прохождению	108
Исключение дублирования кода	108
Программирование по разности	110
Резюме	118

Глава 9. Класс нельзя ввести в средства тестирования	119
Пример раздражающего параметра	119
Пример скрытой зависимости	126
Пример пятна построения объекта	129
Пример раздражающей глобальной зависимости	131
Пример ужасных зависимостей включения	139
Пример многослойного параметра	142
Пример совмещенного параметра	144
Глава 10. Метод нельзя выполнить в средствах тестирования	149
Пример скрытого метода	149
Пример “полезного” свойства языка	153
Пример необнаруживаемого побочного эффекта	155
Глава 11. Требуется изменения в коде, но неизвестно, какие методы следует тестировать	163
Осмысление воздействий	163
Осмысление в прямом направлении	169
Распространение воздействий	173
Инструментальные средства для осмысления воздействий	175
Что дает анализ воздействий	177
Упрощение эскизов воздействий	178
Глава 12. На одном участке требуется внести много изменений, но следует ли разрывать зависимости со всеми классами, имеющими к этому отношение	181
Точки пересечения	182
Простой пример	182
Точки пересечения более высокого уровня	185
Оценка структуры кода по точкам сужения	188
Скрытые препятствия, которые таят в себе точки сужения	190
Глава 13. В код требуется внести изменения, но неизвестно, какие тесты писать	191
Характеристические тесты	192
Выяснение характерных особенностей классов	195
Нацеленное тестирование	196
Эвристический анализ для написания характеристических тестов	200
Глава 14. Убийственная зависимость от библиотек	201
Глава 15. Приложение состоит из сплошных вызовов интерфейса API	203
Глава 16. Код недостаточно понятен для его изменения	211
Составление примечаний и эскизов	211
Разметка листингов	212
Разделение ответственности	213

Уяснение структуры метода	213
Извлечение методов	213
Уяснение воздействий в связи с изменением	213
Черновая реорганизация кода	214
Удаление ненужного кода	214
Глава 17. У приложения отсутствует структура	215
Описание системы	216
Открытое событийное взаимодействие классов	219
Особое внимание к обсуждению проекта	222
Глава 18. Когда тестовый код мешает	223
Условные обозначения имен классов	223
Расположение тестов	224
Глава 19. Как благополучно изменить процедурный код	227
Простой случай	227
Тяжелый случай	228
Ввод нового поведения	232
Использование преимуществ ООП	234
Объектно-ориентированный характер процедурного кода	237
Глава 20. Класс слишком крупный и его дальнейшее укрупнение нежелательно	241
Выявление видов ответственности	244
Другие способы	256
Дальнейшие действия	256
Стратегия	256
Тактика	257
Действия после извлечения класса	259
Глава 21. Изменение одного и того же кода повсеместно	261
Первые шаги	264
Глава 22. Необходимо изменить гигантский метод, но нельзя написать для него тест	279
Разновидности гигантских методов	279
Методы без отступов	280
Запутанные методы	281
Обращение с гигантскими методами при поддержке автоматической реорганизации кода	284
Трудности реорганизации кода вручную	286
Внедрение переменной распознавания	287
Извлечение только того, что хорошо известно	290
Подбирание зависимостей	292
Вынос объекта метода	293
Стратегия обращения с гигантскими методами	293
Скелетное представление методов	293

Поиск последовательностей	294
Извлечение в текущий класс	295
Извлечение небольшими фрагментами	295
Будьте готовы к повторному извлечению	295
Глава 23. Как узнать, нарушают ли что-нибудь изменения в коде	297
Сверхосторожная правка	297
Правка с единственной целью	298
Сохранение сигнатур	300
Упор на компилятор	302
Парное программирование	303
Глава 24. Сдаемся — дальнейшее улучшение невозможно	305
Часть III. Методы разрыва зависимостей	307
Глава 25. Способы разрыва зависимостей	309
Адаптация параметра	309
Процедура	312
Вынос объекта метода	313
Процедура	318
Расширение определения	318
Процедура	320
Инкапсуляция глобальных ссылок	320
Процедура	325
Раскрытие статического метода	325
Процедура	328
Извлечение и переопределение вызова	328
Процедура	330
Извлечение и переопределение фабричного метода	330
Процедура	331
Извлечение и переопределение получателя	332
Процедура	334
Извлечение средства реализации	335
Процедура	337
Более сложный пример	338
Извлечение интерфейса	339
Процедура	343
Ввод делегата экземпляра	345
Процедура	346
Ввод статического установщика	347
Процедура	351
Подстановка связи	352
Процедура	352
Параметризация конструктора	353
Процедура	356
Параметризация метода	356

Процедура	357
Примитивизация параметра	357
Процедура	359
Вытягивание свойства	360
Процедура	363
Вытеснение зависимости	363
Процедура	366
Замена функции указателем функции	366
Процедура	369
Замена глобальной ссылки получателем	369
Процедура	371
Подклассификация и переопределение метода	371
Процедура	373
Замена переменной экземпляра	374
Процедура	377
Переопределение шаблона	377
Процедура	380
Переопределение исходного текста	380
Процедура	381
Приложение. Реорганизация кода	383
Извлечение метода	383
Словарь специальных терминов	387
Предметный указатель	389

*Посвящается Энн, Деборе и Райану —
самым ярким средоточиям моей жизни.*

— Майкл

Предисловие

“С этого все и началось...” — этой фразой Майкл Физерс описывает во вступлении к своей книге момент, с которого началось его страстное увлечение программированием.

“С этого все и началось...” Знакомо ли вам подобное ощущение? Можете ли вы определить такой момент в своей жизни, когда для вас “все и началось”? Было ли это отдельное событие, круто изменившее течение вашей жизни и приведшее в конечном итоге к тому, что вы приобрели эту книгу и начали ее чтение с настоящего предисловия?

Со мной это случилось в шестом классе школы. Я заинтересовался наукой, техникой и космосом. И тогда моя мать нашла и заказала для меня по каталогу детский пластмассовый конструктор для сборки игрушечного компьютера. Он назывался *Digi-Comp I*. Сорок лет спустя этот пластмассовый конструктор занимает почетное место на моей книжной полке. Он послужил тем стимулом, который пробудил во мне неугасающий до сих пор интерес к программированию. Благодаря ему я получил первое, хотя и весьма отдаленное представление о том, насколько интересно писать программы, способные решать задачи для людей. Тот игрушечный компьютер состоял всего лишь из трех RS-триггеров и шести схем И в виде пластмассовых конструкций, но и этого было достаточно, чтобы научиться азам программирования. С того все для меня и началось...

Но мой пыл вскоре был умерен осознанием того факта, что программные системы практически всегда приходят в полный беспорядок. То, что начинается с ясно выкристаллизовавшегося замысла в уме программиста, со временем загнивает, как кусок испорченного мяса. Небольшая изящная система, построенная год назад, превращается в следующем году в непроходимую трясиину из запутанных функций и переменных.

Почему это происходит? Почему программные системы начинают портиться и почему они не могут работать четко?

В одних случаях мы виним во всем заказчиков, а в других — обвиняем их в изменении требований. Ведь нам удобно верить в то, что если бы заказчики удовлетворились своими требованиями к проектированию программной системы, то она и не давала бы сбоев. Всю вину за изменение требований к системе мы привыкли сваливать на заказчиков.

Итак, на повестке дня стоит следующий актуальный вопрос: *изменение требований*. Проектные решения, не предусматривающие изменение требований, изначально считаются неудачными. В связи с этим цель всякого компетентного разработчика программного обеспечения — выработать такое проектное решение, которое допускало бы изменение требований.

На первый взгляд, решить такую задачу не так-то просто. В самом деле, практически каждая проектируемая система страдает недостатком — медленным, разрушительным загниванием. И это загнивание настолько распространено, что для обозначения испорченных программ был придуман специальный термин. Такие программы мы называем *унаследованным кодом*.

Унаследованный код. Это выражение инстинктивно вызывает отвращение у всякого, занимающегося программированием. Ведь воображение тотчас рисует ужасные картины продиранья сквозь непроходимые таежные буреломы и топи с упорно цепляющимися за ноги ветками деревьев и кустарников, безжалостно впивающимися в тело тучами мошкеры и одуряющими запахами гниения, разложения и застоя. Мучения, которые доставляет разбирательство с унаследованным кодом, нередко гасят любые первые порывы энтузиазма, возникающие при программировании.

Многие из нас пытались найти способы, *препятствующие* устареванию кода. На тему принципов, шаблонов и практических приемов, помогающих программистам поддерживать работоспособность разрабатываемых ими систем, написано немало литературы. Но у Майкла Физерса сложилось по этому поводу совсем иное представление, которого многим из нас недостает: препятствовать устареванию кода совершенно недостаточно. Ведь даже самая дисциплинированная группа разработчиков, владеющая самыми совершенными принципами, пользующаяся самыми лучшими шаблонами и придерживающаяся самых выверенных на практике приемов, время от времени вносит путаницу, которая приводит к постепенной порче и загниванию кода. Воспрепятствовать загниванию недостаточно — следует попытаться *обратить* этот процесс вспять.

Именно такому обращению загнивания вспять и посвящена эта книга. В ней идет речь о том, как запутанная, непонятная и сложная система медленно, но верно, постепенно и по частям превращается в простую, изящно структурированную и правильно спроектированную систему. Это книга об обращении вспять процесса энтропии.

Но предупреждая ваш порыв чрезмерного энтузиазма, должен заметить, что обращение загнивания кода вспять — дело непростое и нескорое. Методы, шаблоны и средства, представленные автором в этой книге, достаточно эффективны, но они требуют усилий, времени, терпения и внимания. Эта книга не является панацеей и не дает никаких рецептов, как исключить сразу все накопившиеся признаки загнивания в вашей системе. Вместо этого в книге описывается ряд дисциплин, концепций и подходов, которые вы можете взять на вооружение и которые *помогут* вам в дальнейшей профессиональной деятельности превратить постепенно *деградирующие* системы в постепенно *совершенствуемые* системы.

— *Роберт К. Мартин*
29 июня 2004 года

Вступление

Помните ли вы первую написанную вами программу? Я помню. Это была небольшая графическая программа, написанная мной для одной из первых моделей ПК. Я начал заниматься программированием позже, чем большинство моих коллег. Разумеется, я знаком с компьютерами с детских лет. Я даже помню свои первые яркие впечатления от мини-компьютера, увиденного в одном учреждении, но в детстве у меня не было возможности даже сесть за компьютер. Позднее, когда я стал подростком, у некоторых из моих сверстников появились первые модели микрокомпьютеров TRS-80. Меня они заинтересовали, хотя я испытывал некоторые опасения, поскольку знал, что компьютерные игры затягивают. Поэтому я старался оставаться безразличным к ним. Даже не знаю почему, но я проявлял сдержанный интерес к компьютерам. Затем, когда я учился в колледже, у моего соседа по комнате в общежитии появился компьютер, и тогда я приобрел компилятор C, чтобы научиться программировать. С этого все и началось. Я проводил целые ночи, опробуя на практике разные идеи и анализируя исходный код в редакторе emacs, входившем в состав компилятора. Это было увлекательное, интересное занятие, которое мне очень нравилось.

Надеюсь, что нечто подобное пришлось испытать и вам, читатель, т.е. радость от того, что удалось сделать нечто полезное и работоспособное на компьютере. Подобное ощущение знакомо практически всем программистам, которых я спрашивал об этом. Именно оно отчасти побудило многих из нас выбрать эту профессию, но испытываем ли мы его теперь, хотя бы иногда, в наших повседневных трудах?

Несколько лет назад я однажды вечером позвонил после работы своему другу Эрику Миду. Я знал, что Эрик только что начал консультировать новую группу разработчиков, и поэтому я спросил его: “Как у них идут дела?” Он ответил: “Они пишут унаследованный код, дружище”. Это был один из тех немногих случаев, когда слова коллеги задели меня за живое. У меня от них внутри буквально все перевернулось. Эрик очень точно выразил словами то ощущение, которое я нередко испытывал при первом посещении групп разработчиков. Они очень старались, но в конце рабочего дня — то ли потому что на них давили сроки и ответственность момента, то ли из-за отсутствия образцов лучшего кода для сравнения с их трудами — многие из них просто начинали писать унаследованный код.

Что же представляет собой унаследованный код? Раньше я пользовался этим понятием без какого-то конкретного определения. А теперь попробуем проанализировать следующее строгое определение этого понятия: унаследованным называется код, полученный от кого-то другого. Это может быть код, приобретенный одной компанией у другой или же унаследованный одной группой разработчиков от другой при переходе к иному проекту, т.е. *унаследованный код* — это чужой код. Но для программистов это понятие имеет намного более широкий смысл. Оно приобрело со временем множество смысловых оттенков и более глубокое значение.

Какие ассоциации вызывает у вас термин *унаследованный код*, когда вы слышите его? Возможно, у вас, как и у меня, возникают мысли о запутанной, непонятной структуре, коде, который требуется изменить, но на самом деле вы его просто не понимаете. Поэтому у вас появляются воспоминания о бессонных ночах, проведенных в попытках ввести в такой код новые свойства, которые, казалось бы, нетрудно было добавить, о состоянии полной деморализации и болезненном ощущении от базы кода, вызывающем полное безразличие у всей группы разработчиков к коду, от которого проще помереть, чем исправить его. Отчасти вы испытываете ощущение безнадежности своих попыток улучшить подобный

код. Вот поэтому приведенное выше определение не имеет никакого отношения к самому понятию унаследованного кода. Ведь код может деградировать разными путями, которые зачастую никак не связаны с местом его происхождения.

В программировании понятие *унаследованный код* нередко употребляется как жаргонное обозначение трудноизменяемого кода, который совершенно непонятен. Но приобретя многолетний опыт работы с группами разработчиков над разрешением серьезных осложнений с кодом, я пришел к другому определению данного понятия.

С моей точки зрения, *унаследованный код* — это просто код, не прошедший тесты. Такое определение далось мне горьким опытом. Что же должны делать тесты для выявления неудачного кода? Для меня ответ на данный вопрос очевиден, и он составляет главный предмет, подробно разрабатываемый в этой книге.

Код без тестов является неудачным. И совершенно неважно, насколько хорошо он написан, объектно-ориентирован или инкапсулирован. С помощью текстов мы можем быстро и под полным контролем изменить поведение нашего кода. А без них мы на самом деле не знаем, становится ли наш код лучше или хуже.

Такое условие может показаться слишком суровым. Как, например, быть с чистым кодом? Если база кода довольно чиста и правильно структурирована, то разве этого недостаточно? Отвечая на этот вопрос, нужно постараться избежать ошибки. Мне лично нравится чистый код — и даже больше, чем многим из тех, кого я знаю, но одной чистоты кода явно недостаточно. Разработчики зачастую идут на большой риск, пытаются радикально изменить код без тестирования. Это все равно, что заниматься воздушной гимнастикой без страховочной сетки. Каждый совершаемый шаг требует поразительного умения и ясного понимания того, что может произойти. Точно знать, что произойдет, если изменить пару переменных, — это зачастую все равно, что быть полностью уверенным в том, что ваш партнер-гимнаст подхватит вас за руки, когда вы совершите очередное сальто в воздухе. Если вы работаете в группе, разрабатывающей совершенно чистый код, значит, вы находитесь в лучшем положении, чем большинство других программистов. Мой опыт показывает, что группы, разрабатывающие весь код такой чистоты, встречаются крайне редко. Они, скорее, исключение из общего правила. И знаете почему? Если у них нет поддерживающих тестов, то изменения в их коде будут вноситься намного медленнее, чем в тех группах, где есть такие тесты.

Конечно, группы могут стремиться разрабатывать чистый код с самого начала, но для того, чтобы сделать чище старый код, им потребуется немало времени. И, как правило, этого не удастся сделать полностью. Именно поэтому я безо всяких колебаний определяю унаследованный код как код, не прошедший тесты. Это, на мой взгляд, удачное и практичное определение, указывающее на решение проблемы подобного рода.

Выше вскользь упоминалось о тестах, но эта книга посвящена не тестам, а методам, позволяющим уверенно вносить изменения в любую базу кода. В последующих главах рассматриваются методы, применяемые для лучшего понимания кода, его тестирования, реорганизации и дополнения новыми свойствами.

Читая эту книгу, вы непременно обратите внимание на то, что в ней не идет речь о совершенно конкретном коде. Примеры кода, приведенные в этой книге, носят весьма условный характер, поскольку я не имею права полностью раскрывать код своих клиентов. Но во многих примерах я постарался сохранить основную суть прикладного кода. Нельзя сказать, что эти примеры всегда характерны и показательны. Это, безусловно, лишь небольшие фрагменты качественного кода, хотя они представляют собой далеко не самое

лучшее из того, что можно было бы использовать в качестве примера в данной книге. Помимо соблюдения условий конфиденциальности по отношению к своим клиентам, я просто не мог привести в этой книге такой код, чтобы не скрыть важные моменты во множестве мелких деталей и тем самым довести вас, читатель, до полного отчаяния. По этим причинам многие примеры кода в этой книге довольно лаконичны. Если при анализе любого из этих примеров вам покажется, что вы применяете намного более крупные методы и получаете гораздо более худшие результаты, то обратитесь к совету, который я даю по тексту, и сами решите, настолько он вам подходит, несмотря на кажущуюся простоту приводимого примера.

Методы, представленные в этой книге, были проверены на достаточно крупных фрагментах кода. Но рамки этой книги просто не позволяют привести в ней более крупные примеры кода. В частности, когда в примерах кода встречаются знаки многоточия (...), то их следует трактовать как указание вставить 500 или около того строк скверного кода, как, например:

```
m_pDispatcher->register(listener);  
...  
m_nMargins++;
```

В этой книге речь идет не только о совершенно конкретном коде, но и о совершенно конкретном проектировании. Качественное проектирование должно стать нашей конечной целью, а унаследованный код — это некий промежуточный результат на пути к этой цели. В некоторых главах этой книги описываются способы ввода нового кода в уже существующую базу кода и показывается, как это делается, исходя из удачно выбранных принципов проектирования. Конечно, вы можете постепенно расширить участки очень хорошего, качественного кода в базе унаследованного кода, но не удивляйтесь, если на определенной стадии внесения изменений код станет скверным. Такая работа сродни хирургическому вмешательству. Нам нужно сделать разрез, добраться до внутренностей и, оставив на время эстетические соображения, ответить на вопрос — можно ли улучшить состояние внутренних органов пациента? Если да, то должны ли мы сразу же зашить пациента, посоветовать ему правильно питаться и бегать на длинные дистанции и тут же забыть о его болезни? Конечно, можно поступить и так, но на самом деле нам нужно объективно оценить состояние здоровья пациента, правильно поставить диагноз его заболевания, постараться вылечить и вернуть пациента в более здоровое состояние. Возможно, он уже не будет отличаться олимпийским здоровьем, но нельзя себе позволить, чтобы лучшее стало врагом хорошего. Базы кода должны стать более здоровыми и простыми в использовании. Если пациент почувствует себя чуть лучше, то это зачастую очень удобный момент, чтобы помочь ему придерживаться более здорового образа жизни. Именно эту цель мы и преследуем в отношении унаследованного кода, т.е. мы пытаемся добиться того момента, когда обычно испытываем облегчение. Мы его ожидаем и активно стремимся упростить изменение кода. А когда мы поддерживаем это ощущение во всей группе разработчиков, то результаты проектирования сразу же улучшаются.

Способы, рассматриваемые в этой книге, выявлены и изучены мной вместе с моими коллегами в течение многолетней работы с клиентами, когда мы пытались установить контроль над непокорными базами кода. С остротой проблемы унаследованного кода я столкнулся совершенно случайно. Когда я только начал сотрудничать с компанией Object Mentor, то основной моей задачей была помощь группам разработчиков для разрешения серьезных трудностей и доведения их квалификации и взаимодействия до такого состояния, когда они могли выдавать качественный код. Мы часто пользовались практическими приемами экстремального программирования, чтобы помочь группам разработчиков луч-

ше контролировать свою работу, интенсивнее сотрудничать друг с другом и добиваться нужного результата. Мне часто кажется, что экстремальное программирование больше подходит для сплочения коллектива разработчиков, которым нужно производить качественное программное обеспечение каждые две недели, чем для разработки самого программного обеспечения.

Но трудности начались с самого начала. Многие проекты экстремального программирования оказались незрелыми. У клиентов, с которыми мне приходилось иметь дело, были необычайно крупные базы кода, что доставляло им немало хлопот. Им нужно было каким-то образом контролировать свою работу и начать выдавать результат. Со временем я обнаружил, что постоянно проделываю со своими клиентами одно и то же. Это ощущение достигло апогея своей остроты во время работы над одним из проектов, который группа консультируемых мной разработчиков выполняла в области финансов. До моего прихода в группу ее члены уже пришли к выводу, что блочное тестирование — отличное средство, но тесты, которые они выполняли, были написаны по полному сценарию, совершали многочисленные обращения к базе данных и исполняли крупные фрагменты кода. Писать такие тесты было нелегко, и группа выполняла их не очень часто, поскольку для этого требовалось немало времени. Как только я принялся разрывать вместе с ними зависимости, чтобы получить более мелкие фрагменты кода для тестирования, меня посетило ужасное и малоприятное ощущение, то я уже проделывал нечто подобное едва ли не с каждой группой разработчиков, которую я консультировал. Это была черновая работа, которую приходится выполнять, чтобы работать с кодом, держа его под постоянным контролем, если, конечно, знать, как это делается. И тогда я решил, что мне стоит поразмыслить над тем, как мы разрешаем подобные затруднения и фиксируем их, чтобы помочь группе разработчиков преодолеть их и упростить ей работу с своей базой кода.

Еще одно замечание относительно примеров кода, приведенных в этой книге: они составлены на разных языках программирования. Большая их часть написана на языках Java, C++ и C. В частности, Java был выбран потому, что это весьма распространенный язык программирования, C++ — поскольку на этом языке можно представить ряд особых трудностей, возникающих в унаследованной среде, а C — из-за того, что он позволяет высветить многие проблемы, характерные для процедурного унаследованного кода. Все эти языки программирования охватывают большую часть спектра вопросов, возникающих при рассмотрении унаследованного кода. Но даже если вы программируете на другом языке, то примеры кода, представленные на упомянутых выше языках, все равно окажутся полезными для вас, и поэтому стоит их проанализировать. Ведь многие методы, рассматриваемые в этой книге, могут с тем же успехом применяться при программировании на других языках, включая Delphi, Visual Basic, COBOL и Fortran.

Надеюсь, что методы, представленные в этой книге, окажутся полезными для вас и позволят вновь обрести утраченный по тем или иным причинам интерес к программированию, которое приносит удовольствие и удовлетворение от проделанной работы. Если же вы не испытываете ни то, ни другое в своей повседневной работе, то я надеюсь, что предлагаемые мной методы помогут вам и вашим коллегам по работе вновь испытать эти ощущения.

Благодарности

Прежде всего, я в серьезном долгу перед своей женой Энн и моими детьми Деборой и Райаном. Подготовка, написание и выход в свет этой книги стали возможными благодаря их любви и моральной поддержки. Мне бы хотелось также поблагодарить “дядю Боба” Мартина, президента и основателя компании Object Mentor, — за строгий прагматичный подход к разработке и проектированию программного обеспечения, позволяющий отделить самое главное от несущественного и давший мне возможность прочно утвердиться в своей профессии почти 10 лет назад, когда мои перспективы казались весьма зыбкими, а также за возможность лучше понимать код и поработать за прошедшие пять лет с таким количеством людей, которого я просто не мог бы себе раньше представить.

Благодарю также Кента Бека (Kent Beck), Мартина Фаулера (Martin Fowler), Рона Джеффриса (Ron Jeffries) и Уарда Каннингхэма (Ward Cunningham) за полезные советы и обучение искусству проектирования, программирования и работы с людьми. Особая благодарность выражается всем, кто рецензировал рукопись этой книги, в том числе официальным рецензентам Свену Гортсу (Sven Gorts), Роберту К. Мартину (Robert C. Martin), Эрику Миду (Erik Meade) и Биллу Уэйку (Bill Wake), а также неофициальным рецензентам: д-ру Роберту Коссу (Dr. Robert Koss), Джеймсу Греннингу (James Grenning), Лоуэллу Линдстрэму (Lowell Lindstrom), Майке Мартину (Micah Martin), Рассу Руферу (Russ Rufer) и сотрудникам компании Silicon Valley Patterns Group и Джеймсу Ньюкирку (James Newkirk).

Благодарю также рецензентов самых первых вариантов рукописи книги, опубликованных в Интернете. Их отзывы оказали существенное влияние на направленность книги после реорганизации ее формата. Заранее приношу извинения тем, кого я здесь не упомянул. Итак, выражаю свою искреннюю признательность следующим первым рецензентам моей книги: Даррену Хоббсу (Darren Hobbs), Мартину Липперту (Martin Lippert), Киту Николасу (Keith Nicholas), Флипу Пламли (Philip Plumlee), К. Киту Рэю (C. Keith Ray), Роберту Блему (Robert Blum), Биллу Беррису (Bill Burris), Уильяму Капуто (William Caputo), Брайану Мэрику (Brian Marick), Стиву Фриману (Steve Freeman), Дэвиду Путмену (David Putman), Эмили Бак (Emily Bache), Дэйву Эстелсу (Dave Astels), Расселу Хиллу (Russel Hill), Кристиану Сепульведе (Christian Sepulveda) и Брайану Кристоферу Робинсону (Brian Christopher Robinson).

Кроме того, выражаю благодарность Джошуа Кериевски (Joshua Kerievsky) — за первое рецензирование книги, а также Джеффу Лангру (Jeff Langr) — за полезные советы и периодическое рецензирование по ходу работы над книгой.

Все рецензенты рукописи моей книги помогли мне значительно улучшить ее качество, и если в ней найдутся ошибки, то только мои собственные.

Благодарю Мартина Фаулера, Ральфа Джонсона (Ralph Johnson), Билла Опдайка (Bill Opdyke), Дона Робертса (Don Roberts) и Джона Бранта (John Brant) за их труды в области реорганизации кода. Они меня вдохновляли.

Кроме того, я в огромном долгу перед Джеем Пакликом (Jay Packlick), Жаком Морелем (Jacques Morel) и Келли Мауэр (Kelly Mower) из фирмы Sabre Holdings, а также перед Грэхемом Райтом (Graham Wright) из компании Workshare Technology за их поддержку и отзывчивость.

Особая благодарность выражается Полу Петралии (Paul Petralia) — за оказанную помощь и своевременную поддержку, так необходимые автору книги, а также Мишель

Винсенти (Michelle Vincenti), Лори Лайонс (Lori Lyons), Кристе Хансинг (Krista Hansing) и остальным сотрудникам издательства Prentice-Hall.

Особая признательность выражается также Гэри (Gary) и Джоан Физерс (Joan Feathers), Эйприл Робертс (April Roberts), д-ру Раймунду Эге (Dr. Raimund Ege), Дэвиду Лопесу де Квинтана (David Lopez de Quintana), Карлосу Пересу (Carlos Perez), Карлосу М. Родригесу (Carlos M. Rodriguez) и почтенному д-ру Джону К. Комфорту (Dr. John C. Comfort) — за многолетнюю поддержку и оказанную помощь. Благодарю и Брайна Баттона (Brian Button) за пример, составленный им к главе 21 этой книги. Он написал код данного примера почти за час, когда мы составляли вместе учебный курс по реорганизации кода, и этот код стал для меня излюбленным учебным примером.

Кроме того, особая благодарность выражается Янику Топу (Janik Top), инструментальная пьеса которого “Будущее” (De Futura) вдохновляла меня в последние недели работы над книгой.

И наконец, мне бы хотелось поблагодарить всех, с кем мне пришлось работать в течение последних нескольких лет, чья пытливость и сомнения только способствовали упрочению материала этой книги.

— Майкл Физерс

mfeathers@objectmentor.com

www.objectmentor.com

www.michaelfeathers.com