ОГЛАВЛЕНИЕ

Предисловие Роберта С. Мартина ко второму изданию 14

Предисловие Майкла Фэзерса ко второму изданию 16

Вступление 18

Благодарности 20

Об этой книге 21

Предполагаемая аудитория 22
Структура книги 22
Графические выделения и загрузка исходного кода 23
Требования к программному обеспечению 24
Автор в сети 24
Другие проекты Роя Ошероува 25

Об иллюстрации на обложке 26

ЧАСТЬ I. Приступая к работе 27

Глава 1. Основы автономного тестирования 28

1.1. Определение автономного тестирования, шаг за шагом ... 29
1.1.1. О важности написания хороших автономных тестов 31
1.1.2. Все мы писали автономные тесты (или что-то в этом роде) ... 31
1.2. Свойства хорошего автономного теста 32
1.3. Интеграционные тесты 33
1.3.1. Недостатки неавтоматизированных интеграционных тестов по сравнению с автоматизированными автономными тестами 36
1.4. Из чего складывается хороший автономный тест? 39
1.5. Пример простого автономного теста 40
1.6. Разработка через тестирование 44
1.7. Три основных навыка успешного практика TDD 47
1.8. Резюме 48

Глава 2. Первый автономный тест..... 50
2.1. Каркасы автономного тестирования 51
2.1.1. Что предлагают каркасы автономного тестирования.....51
2.1.2. Каркасы семейства xUnit.....54
2.2. Знакомство с проектом LogAn..... 54
2.3. Первые шаги освоения NUnit..... 55
2.3.1. Установка NUnit.....55
2.3.2. Загрузка решения 57
2.3.3. Использование атрибутов NUnit 60
2.4. Создание первого теста..... 61
2.4.1. Класс Assert 62
2.4.2. Прогон первого теста в NUnit 63
2.4.3. Добавление положительных тестов.....64
2.4.4. От красного к зеленому: тесты должны проходить.....65
2.4.5. Стилистическое оформление тестового кода.....66
2.5. Рефакторинг – параметризованные тесты 67
2.6. Другие атрибуты в NUnit..... 69
2.6.1. Подготовка и очистка 70
2.6.2. Проверка ожидаемых исключений.....73
2.6.3. Игнорирование тестов 76
2.6.4. Текучий синтаксис в NUnit 77
2.6.5. Задание категорий теста.....77
2.7. Проверка изменения состояния системы,
а не возвращаемого значения 78
2.8. Резюме 83

ЧАСТЬ II.
Основные приемы 85

Глава 3. Использование заглушек для разрыва зависимостей 86
3.1. Введение в заглушки..... 86
3.2. Выявление зависимости от файловой системы в LogAn 88
3.3. Как можно легко протестировать LogAnalyzer..... 89
3.4. Рефакторинг проекта с целью повышения
тестопригодности 92
3.4.1. Выделение интерфейса с целью подмены истинной
реализации 93
3.4.2. Внедрение зависимости: внедрение поддельной
реализации в тестируемую единицу работы 96
3.4.3. Внедрение подделки на уровне конструктора
(внедрение через конструктор) 97
3.4.4. Имитация исключений от подделок.....101
3.4.5. Внедрение подделки через установку свойства 102

3.4.6. Внедрение подделки непосредственно перед вызовом метода ... 104
3.5. Варианты рефакторинга ... 112
3.5.1. Использование выделения и переопределения для создания поддельных результатов ... 113
3.6. Преодоление проблемы нарушения инкапсуляции ... 115
3.6.1. internal и [InternalsVisibleTo] ... 116
3.6.2. Атрибут [Conditional] ... 116
3.6.3. Использование директив #if и #endif для условной компиляции ... 117
3.7. Резюме ... 118

Глава 4. Тестирование взаимодействий с помощью подставных объектов ... 120

4.1. Сравнение тестирования взаимодействий с тестированием на основе значений и состояния ... 121
4.2. Различия между подставками и заглушками ... 124
4.3. Пример простой рукописной подставки ... 126
4.4. Совместное использование заглушки и подставки ... 128
4.5. Одна подставка на тест ... 134
4.6. Цепочки подделок: заглушки, порождающие подставки или другие заглушки ... 135
4.7. Проблемы рукописных заглушек и подставок ... 136
4.8. Резюме ... 137

Глава 5. Изолирующие каркасы генерации подставных объектов ... 139

5.1. Зачем использовать изолирующие каркасы? ... 140
5.2. Динамическое создание поддельного объекта ... 142
5.2.1. Применение NSubstitute в тестах ... 143
5.2.2. Замена рукописной подделки динамической ... 144
5.3. Подделка значений ... 147
5.3.1. Встретились в тесте подставка, заглушка и священник ... 148
5.4. Тестирование операций, связанных с событием ... 154
5.4.1. Тестирование прослушивателя события ... 154
5.4.2. Тестирование факта генерации события ... 156
5.5. Современные изолирующие каркасы для .NET ... 157
5.6. Достоинства и подводные камни изолирующих каркасов ... 159
5.6.1. Каких подводных камней избегать при использовании изолирующих каркасов ... 159
5.6.2. Неудобочитаемый тестовый код ... 160
5.6.3. Проверка не того, что надо ... 160
5.6.4. Наличие более одной подставки в одном тесте ... 160

5.6.5. Избыточное специфицирование теста 161
5.7. Резюме 162

Глава 6. Внутреннее устройство изолирующих каркасов 164

6.1. Ограниченные и неограниченные каркасы 164
6.1.1. Ограниченные каркасы 165
6.1.2. Неограниченные каркасы 165
6.1.3. Как работают неограниченные каркасы на основе профилировщика 168
6.2. Полезные качества хороших изолирующих каркасов 170
6.3. Особенности, обеспечивающие неустареваемость и удобство пользования 171
6.3.1. Рекурсивные подделки 172
6.3.2. Игнорирование аргументов по умолчанию 173
6.3.3. Массовое подделывание 173
6.3.4. Нестрогое поведение подделок 174
6.3.5. Нестрогие подставки 175
6.4. Антипаттерны проектирования в изолирующих каркасах 175
6.4.1. Смешение понятий 176
6.4.2. Запись и воспроизведение 177
6.4.3. Липкое поведение 178
6.4.4. Сложный синтаксис 179
6.5. Резюме 180

ЧАСТЬ III. Тестовый код 181

Глава 7. Иерархии и организация тестов 182

7.1. Прогон автоматизированных тестов в ходе автоматизированной сборки 183
7.1.1. Анатомия скрипта сборки 184
7.1.2. Запуск сборки и интеграции 186
7.2. Распределение тестов по скорости и типу 188
7.2.1. Разделение автономных и интеграционных тестов и человеческий фактор 189
7.2.2. Безопасная зеленая зона 190
7.3. Тесты должны храниться в системе управления версиями 191
7.4. Соответствие между тестовыми классами и тестируемым кодом 191
7.4.1. Соответствие между тестами и проектами 192
7.4.2. Соответствие между тестамии классами 192

7.4.3. Соответствие между тестами и точками входа в единицу работы ... 194
7.5. Внедрение сквозной функциональности ... 194
7.6. Разработка API тестов приложения ... 197
7.6.1. Наследование тестовых классов ... 197
7.6.2. Создание служебных классов и методов для тестов ... 212
7.6.3. Извещение разработчиков об имеющемся API ... 213
7.7. Резюме ... 214

Глава 8. Три столпа хороших автономных тестов ... 216

8.1. Написание заслуживающих доверия тестов ... 217
8.1.1. Когда удалять или изменять тесты ... 217
8.1.2. Устранение логики из тестов ... 223
8.1.3. Тестирование только одного результата ... 225
8.1.4. Разделение автономных и интеграционных тестов ... 227
8.1.5. Проводите анализ кода, уделяя внимание покрытию кода ... 227
8.2. Написание удобных для сопровождения тестов ... 230
8.2.1. Тестирование закрытых и защищенных методов ... 230
8.2.2. Устранение дублирования ... 233
8.2.3. Применение методов подготовки без усложнения сопровождения ... 237
8.2.4. Принудительная изоляция тестов ... 240
8.2.5. Предотвращение нескольких утверждений о разных функциях ... 247
8.2.6. Сравнение объектов ... 250
8.2.7. Предотвращение избыточного специфицирования ... 253
8.3. Написание удобочитаемых тестов ... 255
8.3.1. Именование автономных тестов ... 256
8.3.2. Именование переменных ... 257
8.3.3. Утверждения со смыслом ... 258
8.3.4. Отделение утверждений от действий ... 259
8.3.5. Подготовка и очистка ... 260
8.4. Резюме ... 261

ЧАСТЬ IV.
Проектирование и процесс ... 263

Глава 9. Внедрение автономного тестирования в организации ... 264

9.1. Как стать инициатором перемен ... 265
9.1.1. Будьте готовы к трудным вопросам ... 265
9.1.2. Убедите сотрудников: сподвижники и противники ... 265
9.1.3. Выявите возможные пути внедрения ... 267
9.2. Пути к успеху ... 269
9.2.1. Партизанское внедрение (снизу вверх) ... 269

9.2.2. Обеспечение поддержки руководства (сверху вниз) 270
9.2.3. Привлечение организатора со стороны 270
9.2.4. Наглядная демонстрация прогресса 271
9.2.5. Постановка конкретных целей 272
9.2.6. Осознание неизбежности препятствий 274
9.3. Пути к провалу 275
9.3.1. Отсутствие движущей силы 275
9.3.2. Отсутствие политической поддержки 275
9.3.3. Плохая организация внедрения и негативные первые впечатления 276
9.3.4. Отсутствие поддержки со стороны команды 276
9.4. Факторы влияния 277
9.5. Трудные вопросы и ответы на них 279
9.5.1. Насколько автономное тестирование замедлит текущий процесс? 280
9.5.2. Не станет ли автономное тестирование угрозой моей работе в отделе контроля качества? 282
9.5.3. Откуда нам знать, что автономные тесты и вправду работают? 282
9.5.4. Есть ли доказательства, что автономное тестирование действительно помогает? 283
9.5.5. Почему отдел контроля качества по-прежнему находит ошибки? 283
9.5.6. У нас полно кода без тестов: с чего начать? 284
9.5.7. Мы работаем на нескольких языках, возможно ли при этом автономное тестирование? 285
9.5.8. А что, если мы разрабатываем программно-аппаратные решения? 285
9.5.9. Откуда нам знать, что в тестах нет ошибок? 285
9.5.10. Мой отладчик показывает, что код работает правильно. К чему мне еще тесты? 286
9.5.11. Мы обязательно должны вести разработку через тестирование? 286
9.6. Резюме 287

Глава 10. Работа с унаследованным кодом 288

10.1. С чего начать добавление тестов? 289
10.2. На какой стратегии выбора остановиться 291
10.2.1. Плюсы и минусы стратегии «сначала простые» 291
10.2.2. Плюсы и минусы стратегии «сначала трудные» 292
10.3. Написание интеграционных тестов до рефакторинга 293
10.4. Инструменты, важные для автономного тестирования унаследованного кода 294
10.4.1. Изолируйте зависимости с помощью JustMock или Typemock Isolator 295
10.4.2. Используйте JMockit при работе с унаследованным кодом на Java 297

10.4.3. Используйте Vise для рефакторинга кода на Java 298
10.4.4. Используйте приемочные тесты перед началом рефакторинга 299
10.4.5. Прочитайте книгу Майкла Фэзерса об унаследованном коде 300
10.4.6. Используйте NDepend для исследования продуктового кода 301
10.4.7. Используйте ReSharper для навигации и рефакторинга продуктового кода 302
10.4.8. Используйте Simian и TeamCity для обнаружения повторяющегося кода (и ошибок) 302
10.5. Резюме 303

Глава 11. Проектирование и тестопригодность 304

11.1. Почему я должен думать о тестопригодности в своем проекте? 304
11.2. Цели проектирования с учетом тестопригодности 305
11.2.1. По умолчанию делайте методы виртуальными 307
11.2.2. Проектируйте на основе интерфейсов 308
11.2.3. По умолчанию делайте классы незапечатанными 308
11.2.4. Избегайте создания экземпляров конкретных классов внутри методов, содержащих логику 308
11.2.5. Избегайте прямых обращений к статическим методам 309
11.2.6. Избегайте конструкторов и статических конструкторов, содержащих логику 309
11.2.7. Отделяйте логику объектов-одиночек от логики их создания 310
11.3. Плюсы и минусы проектирования с учетом тестопригодности 311
11.3.1. Объем работы 313
11.3.2. Сложность 313
11.3.3. Раскрытие секретной интеллектуальной собственности 314
11.3.4. Иногда нет никакой возможности 314
11.4. Альтернативы проектированию с учетом тестопригодности 314
11.4.1. К вопросу о проектировании в динамически типизированных языках 315
11.5. Пример проекта, трудного для тестирования 317
11.6. Резюме 321
11.7. Дополнительные ресурсы 322

ПРИЛОЖЕНИЕ. Инструменты и каркасы 325

A.1. Изолирующие каркасы 325
A.1.1. Moq 326

A.1.2. Rhino Mocks 326
A.1.3. Typemock Isolator 327
A.1.4. JustMock 328
A.1.5. Microsoft Fakes (Moles) 328
A.1.6. NSubstitute 329
A.1.7. FakeItEasy 329
A.1.8. Foq 329
A.1.9. Isolator++ 330
A.2. Каркасы тестирования 330
A.2.1. Непрерывный исполнитель тестов Mighty Moose (он же ContinuousTests) 331
A.2.2. Непрерывный исполнитель тестов NCrunch 331
A.2.3. Исполнитель тестов Typemock Isolator 332
A.2.4. Исполнитель тестов CodeRush 332
A.2.5. Исполнитель тестов ReSharper 332
A.2.6. Исполнитель TestDriven.NET 333
A.2.7. Исполнитель NUnit GUI 334
A.2.8. Исполнитель MSTest 334
A.2.9. Pex 335
A.3. API тестирования 335
A.3.1. MSTest API – каркас автономного тестирования от Microsoft 335
A.3.2. MSTest для приложений Metro (магазин Windows) 336
A.3.3. NUnit API 336
A.3.4. xUnit.net 337
A.3.5. Вспомогательный API Fluent Assertions 337
A.3.6. Вспомогательный API Shouldly 337
A.3.7. Вспомогательный API SharpTestsEx 338
A.3.8. Вспомогательный API AutoFixture 338
A.4. IoC-контейнеры 338
A.4.1. Autofac 340
A.4.2. Ninject 340
A.4.3. Castle Windsor 340
A.4.4. Microsoft Unity 340
A.4.5. StructureMap 341
A.4.6. Microsoft Managed Extensibility Framework 341
A.5. Тестирование работы с базами данных 341
A.5.1. Использование интеграционных тестов для уровня данных 342
A.5.2. Использование TransactionScope для отката изменений данных 342
A.6. Тестирование веб-приложений 344
A.6.1. Ivonna 344
A.6.2. Тестирование веб-приложений в Team System 344
A.6.3. Watir 345
A.6.4. Selenium WebDriver 345
A.6.5. Coypu 345

A.6.6. Capybara 345
A.6.7. Тестирование JavaScript 346
A.7. Тестирование пользовательского интерфейса (персональных приложений) 346
A.8. Тестирование многопоточных приложений 347
A.8.1. Microsoft CHESS 347
A.8.2. Osherove.ThreadTester 347
A.9. Приемочное тестирование 348
A.9.1. FitNesse 348
A.9.2. SpecFlow 348
A.9.3. Cucumber 349
A.9.4. TickSpec 349
A.10. Каркасы с API в стиле BDD 349

Предметный указатель 351

ПРЕДИСЛОВИЕ РОБЕРТА С. МАРТИНА КО ВТОРОМУ ИЗДАНИЮ

Было это году в 2009. Я выступал на конференции норвежских разработчиков в Осло (ах, Осло в июне!). Мероприятие проводилось на огромной спортивной арене. Организаторы разделили трибуны на секции, установили перед каждой секцией помосты и развесили между ними черные полотнища, так что получилось восемь отдельных «залов» для заседаний. Помню, я как раз заканчивал доклад, который был посвящен TDD[1], а, может, SOLID[2] или астрономии или еще чему-то, когда внезапно с соседнего помоста раздалось хриплое пение, сопровождаемое громкими гитарными аккордами.

Драпировки не мешали мне взглянуть, что там происходит, и я увидел на соседней сцене парня, который производил весь этот шум. Разумеется, это был Рой Ошероув.

Те из вас, кто меня знает, в курсе, что я тоже в принципе могу под настроение запеть посредине технического доклада о софте. Поэтому, вернувшись к своей аудитории, я подумал, что мы с этим Ошероувом – родственные души, и надо бы познакомиться поближе.

Так я и поступил. И надо сказать, он внес немалый вклад в мою последнюю книгу «The Clean Coder» и три дня вместе со мной вел семинар по TDD. Мой опыт общения с Роем оказался исключительно приятным, надеюсь, что мы еще не раз встретимся.

Уверен, что и ваш опыт общения с Роем – путем прочтения этой книги – будет не менее приятным, поскольку эта книга – нечто особенное.

---

1 Test-driven development – разработка через тестирование. – *Прим. перев.*

2 Single responsibility, Open-closed, Liskov substitution, Interface segregation и Dependency inversion – принципы объектно-ориентированного проектирования: единственной обязанности, подстановки Лисков, разделения интерфейсов и инверсии зависимости. – *Прим. перев.*

Вы когда-нибудь читали романы Миченера[3]? Я – нет, но говорят, что все они начинаются «с атома». Книга, которую вы держите в руках, – не роман Джеймса Миченера, но тоже начинается с атома – атома автономного тестирования.

Не впадайте в заблуждение, пролистывая первые страницы. Это не *просто* введение в автономное тестирование. Это лишь начало и, если вы – опытный разработчик, то первые главы можете проглядеть по диагонали. Но из последующих глав, опирающихся друг на друга, будет возведена конструкция, поражающая своей глубиной. Честно скажу, когда я читал последнюю главу (еще не зная, что она последняя), то думал, что в следующей речь пойдет о мире во всем мире – ну действительно, о чем еще говорить, после того как решена проблема внедрения автономного тестирования в упрямо сопротивляющихся организациях с унаследованными системами?

Эта книга техническая – очень техническая. В ней уйма кода. И это хорошо. Но Рой не ограничивается одними лишь техническими проблемами. Иногда он достает гитару и разражается песней – рассказывает случаи из своей практики или пускается в философские рассуждения о смысле проектирования или о том, что такое интеграция. Похоже, ему доставляет искреннее удовольствие потчевать нас историями о грубейших ошибках, которые он совершал в далеком 2006 году.

Да, кстати, не переживайте по поводу того, что все примеры написаны на C#. Я хочу сказать, что по существу-то C# ничем не отличается от Java. Правда? Да и вообще это не имеет значения. Для формулирования своих мыслей Рой может использовать C#, но уроки, извлекаемые из этой книги, в равной мере применимы к Java, C, Ruby, Python, PHP и любому другому языку программированию (за исключением разве что COBOL).

Неважно, кто вы: новичок, еще незнакомый с автономным тестированием и методикой разработки через тестирование или, ас, поднаторевший в этом деле, – что-то для себя в этой книге вы обязательно найдете. Так что готовьтесь с удовольствием послушать, как Рой будет исполнять песню «Искусство автономного тестирования».

Да, Рой, а ты, пожалуйста, настрой уже эту гитару!

Роберт С. Мартин (дядюшка Боб)
CLEANCODER.COM

---

[3] Джеймс Элберт Миченер – американский писатель, автор более 40 произведений, в основном исторических саг, описывающих жизнь нескольких поколений в какомлибо определённом географическом месте. Отличался тщательной проработкой деталей в своих произведениях. – *Прим. перев.*

ПРЕДИСЛОВИЕ МАЙКЛА ФЭЗЕРСА КО ВТОРОМУ ИЗДАНИЮ

Когда Рой Ошероув сообщил мне, что работает над книгой об автономном тестировании, я испытал огромную радость. Идея тестирования витала в отрасли уже много лет, но в материалах, посвященных автономному тестированию, все же ощущался недостаток. На моей книжной полке стояли книги о тестировании вообще и о разработке через тестирование в частности, но до сих пор не было исчерпывающего справочника по автономному тестированию – книги, в котором тема раскрывалась бы с азов и до общепринятых практических приемов. И это поистине удивительно. Ведь автономное тестирование – не новая концепция. Как же мы дошли до жизни такой?

Высказывание о том, что наша отрасль еще очень молода, стало уже чуть ли не общим местом. Но это правда. Не прошло еще и ста лет, как математики заложили основы нашей отрасли, а оборудование, достаточно быстрое, чтобы воплотить их идеи в жизнь, создано лишь 60 лет назад. В нашей отрасли изначально существовал разрыв между теорией и практикой, и только сейчас мы начинаем осознавать, как это отразилось на ней.

Когда-то давно машинное время стоило дорого. Мы запускали пакетные программы. Программистам выделялось время для прогона их задач, они набивали программы на перфокартах и относили колоды в машинный зал. Если в программе была ошибка, то вы теряли выделенное вам время, поэтому приходилось, сидя за столом, мысленно на бумаге проигрывать все возможные сценарии, все граничные случаи. Сомневаюсь, что тогда сама идея автоматизированного автономного тестирования кому-то могла прийти в голову. Зачем использовать машину для тестирования, когда можно задействовать ее для решения задач, ради чего она и была построена? Из-за скудости ресурсов мы пребывали во мраке.

Позже, когда машины стали быстрее, мы отравились ядом интерактивных вычислений. Мы могли вводить и изменять код по собственной прихоти. Идея проверки кода за столом ушла в прошлое, и мы растеряли дисциплину прежних лет. Мы знали, что программировать трудно, но это означало лишь, что мы должны проводить больше времени за компьютером, меняя линии и символы, пока не найдем нужное заклинание.

Мы перешли от скудости сразу к изобилию, проскочив промежуточные этапы, но теперь исправляем это упущение. Автоматизированное автономное тестирование сочетает дисциплину проверки за столом с новым представлением о компьютере, как о ресурсе для разработки. Мы можем писать автоматизированные тесты на том же языке, на котором написана сама программа, чтобы контролировать свою работу – не однократно, а так часто, как способны эти тесты прогонять. Не думаю, что в разработке программного обеспечения есть еще какая-нибудь столь же действенная практика.

Сейчас, в 2009 году, когда я пишу эти строки, книга Роя уже передана в производство, и я очень рад этому. Она являет собой практическое руководство, которое поможет вам на первых шагах и будет служить отличным справочником по решению связанных с тестированием задач. «Искусство автономного тестирования» – не книга об идеализированных сценариях. Она научит вас, как тестировать реальный код, как пользоваться популярными каркасами и, самое главное, как писать код, удобный для тестирования. Книга с таким названием должна была бы появиться много лет назад, но тогда мы не были к ней готовы. Теперь готовы. Радуйтесь.

Майкл Фезерс

ВСТУПЛЕНИЕ

В одном из самых крупных провалившихся проектов, над которыми я работал, автономные тесты были. Или мне так казалось. Я возглавлял группу программистов, писавших приложение для выставления счетов, и разрабатывали мы, как положено, через тестирование – писали сначала тест, потом код, наблюдали, как тест не проходит, изменяли код, чтобы тест прошел, производили рефакторинг – и все по новой.

Первые несколько месяцев все шло как по маслу. У нас были тесты, доказывавшие, что код работает. Но со временем требования изменялись. И мы были вынуждены изменять код в соответствии с новыми требованиями, а при этом переставали работать тесты, и их тоже приходилось исправлять. Код все еще работал, но написанные нами тесты стали такими хрупкими, что малейшее изменение в коде приводило к их отказу, хотя сам код работал правильно. Задача модификации кода в классе или методе обернулась сущим кошмаром, так как необходимо было править все сопутствующие автономные тесты.

Хуже того, некоторые тесты стали непригодны, потому что писавшие их люди уволились, и никто не знал, как эти тесты сопровождать и что вообще они проверяют. Имена наших методов автономного тестирования оказались недостаточно понятными, а некоторые тесты зависели от других. В конце концов, не прошло и шести месяцев, как мы выбросили большую часть тестов.

Проект с треском провалился, потому что мы довели дело до того, что тесты приносили больше вреда, чем пользы. На их сопровождение и понимание уходило больше времени, чем экономилось, поэтому мы перестали их использовать. Впоследствии при работе над другими проектами мы уже подходили к автономным тестам более обдуманно, и это принесло свои плоды, позволив сэкономить кучу времени на отладке и интеграции. Но со времен того первого неудачного проекта я собираю наиболее удачные приемы автономного тестирования и применяю их в последующих проектах. Каждый новый проект пополняет эту копилку.

Именно рассказу о том, как писать автономные тесты – и при этом делать их удобными для сопровождения и восприятия, а также достойными доверия, – и посвящена эта книга. Ни язык, ни интегрированная среда разработки (IDE) значения не имеют. В начале мы рассмотрим основы создания автономного теста, затем перейдем к тестированию взаимодействий и к изложению рекомендаций по написанию, управлению и сопровождению автономных тестов в реальных условиях.

БЛАГОДАРНОСТИ

Выражаю огромную благодарность Майклу Стивенсу (Michael Stephens) и Нермине Миллер (Nermina Miller) из издательства Manning, которые терпеливо сопровождали меня на всех этапах долгого пути по написанию этой книги. Спасибо также всем сотрудникам издательства, которые работали над выпуском второго издания, иногда оставаясь невидимыми для меня.

Спасибо Джиму Ньюкирку (Jim Newkirk), Майклу Фэзерсу (Michael Feathers), Джерарду Мезарошу (Gerard Meszaros) и многим другим, у кого я черпал вдохновение и идеи, составившие эту книгу. Отдельное спасибо дядюшке Бобу Мартину за то, что он согласился написать предисловие ко второму изданию.

Хочу также поблагодарить за ценные отзывы рецензентов, которые читали рукопись на различных этапах подготовки: Аарон Колкорд (Aaron Colcord), Алессандро Кампеизм (Alessandro Campeism), Алессандро Галло (Alessandro Gallo), Билл Соренсен (Bill Sorensen), Бруно Соннино (Bruno Sonnino), Камаль Чакар (Camal Cakar), Дэвид Мадурос (David Madouros), д-р Франсиш Буонтемпо (Frances Buontempo), Дрор Хэлпер (Dror Helper), Франческо Гогги (Francesco Goggi), Иван Пазминьо (Iván Pazmiño), Джейсон Хэйлз (Jason Hales), Джуао Ангело (João Angelo), Калеб Педерсон (Kaleb Pederson), Карл Метивир (Karl Metivier), Мартин Скурла (Martin Skurla), Мартин Флэтчер (Martyn Fletcher), Пол Стэк (Paul Stack), Филипп Ли (Philip Lee), Прадип Челлаппан (Pradeep Chellappan), Рафаэль Фариа (Raphael Faria), Тим Слоун (Tim Sloan).

Я признателен также Рикарду Нильсону (Rickard Nilsson) за техническую редактуру окончательного варианта рукописи непосредственно перед сдачей в печать.

И напоследок хочу сказать спасибо читателям ранних вариантов книги, публикуемых по программе предварительного ознакомления издательства Manning, за комментарии в сетевом форуме. Вы помогли книге стать такой, какая она есть.

ОБ ЭТОЙ КНИГЕ

Из всего, что я слышал об обучении, пожалуй, самым умным был совет: если хочешь чему-то по-настоящему научиться, начни это преподавать (не помню, кто это сказал). Работа над первым изданием этой книги, которое вышло в 2009, стала для меня именно таким опытом изучения. Я начал писать книгу, потому что устал снова и снова отвечать на одни и те же вопросы. Но были и другие причины. Я хотел попробовать что-то новое; я хотел поставить эксперимент; мне было интересно, чему я смогу научиться, работая над книгой – любой книгой. Автономное тестирование было предметом, в котором я хорошо разбирался. По крайней мере, я так думал. Беда в том, что чем больше опыта, тем глупее себя ощущаешь.

В первом издании есть места, с которыми я сегодня не согласен – например, что понятие «автономная единица» (unit) относится к методу. Это совершенно неверно. Автономная единица – это единица работы, об этом я говорю в первой главе второго издания. Она может быть совсем мелкой – как метод – или достаточно крупной – несколько классов (а то и сборок). Есть и другие изменения, о которых вы в свое время узнаете.

Что нового во втором издании

Во второе издание я добавил материал об ограниченных и неограниченных изолирующих каркасах, а также новую главу 6 о том, что считать хорошим изолирующим каркасом, и о внутреннем устройстве каркасов типа Typemock.

Я больше не использую RhinoMocks. Держитесь от него подальше. Этот продукт мертвый – по крайней мере, в данный момент. Основы изолирующих каркасов я демонстрирую на примере NSubstitute и рекомендую также FakeItEasy. Я по-прежнему не в восторге от MOQ – по причинам, которые объясняются в главе 6.

Я включил несколько новых приемов в главу о внедрении автономного тестирования на уровне организации.

В код примеров внесено множество проектных изменений. Я почти полностью отказался от установки свойств и использую главным

образом внедрение зависимости через конструктор. Добавлено рассмотрение принципов SOLID, но лишь в объеме, достаточном, чтобы разжечь в вас интерес к самостоятельному изучению этой темы.

В разделах главы 7, относящихся к сборке, тоже есть новая информация. За время, прошедшее с выхода первого издания, я много узнал об автоматизации сборки и соответствующих паттернах.

Я не рекомендую использовать методы подготовки и показываю, как можно реализовать ту же функциональность по-другому. Я также использую последние версии NUnit, поэтому применяемый в книге NUnit API частично изменился.

Изменению подверглись средства, относящиеся к унаследованному коду, описанные в главе 10.

Последние три года я работал не только с .NET, но и с Ruby, и это легло в основу новых соображений о проектировании и тестопригодности, которые я излагаю в главе 10. Все материалы в приложении об инструментах и каркасах актуализированы, сведения об устаревших инструментах исключены.

Предполагаемая аудитория

Эта книга для всех, кто пишет код и хочет узнать о передовой практике автономного тестирования. Все примеры написаны на C# с использованием Visual Studio, поэтому работающим на платформе .NET они будут особенно полезны. Но приведенные рекомендации равным образом относятся к большинству, если не ко всем объектно-ориентированным, статически типизированным языкам (в частности, VB.NET, Java, and C++). Если вы архитектор, разработчик, руководитель группы, инженер по контролю качеству (пишущий код) или начинающий программист, то эта книга для вас.

Структура книги

Если вы никогда не писали автономных тестов, то лучше читать книгу от корки до корки, чтобы составить полную картину. Ну а те, кто уже имеет опыт, могут читать главы выборочно в любом удобном порядке. Книга состоит из четырех частей.

В первой части вы научитесь основам написания автономных тестов: узнаете, как работать с каркасом тестирования (NUnit) и что такое атрибуты автоматизированного тестирования, например `[Test]` и `[TestCase]`. Здесь же рассказывается об утверждениях, игнорировании некоторых тестов, тестировании единицы работы, трех типах

значений, возвращаемых автономным тестом, и трех соответствующих им типах тестов: тесты, основанные на значениях, тесты, основанные на состоянии, и тесты взаимодействия.

Во второй части рассматриваются приемы разрыва зависимостей: подставные объекты, заглушки, изолирующие каркасы и соответствующие им способы рефакторинга кода. В главе 3 вводится понятие о заглушках и показывается, как их вручную создавать и использовать. В главе 4 дается представление о тестировании взаимодействия с помощью написанных вручную подставных объектов. В главе 5 обе идеи сводятся вместе и демонстрируется, как изолирующие каркасы позволяют их объединить и автоматизировать. В главе 6 содержится углубленное обсуждение ограниченных и неограниченных изолирующих каркасов и их внутреннего устройства.

Третья часть посвящена различным способам организации тестового кода, приемам его запуска и переработки структуры, а также передовым методам написания тестов. В главе 7 обсуждаются иерархии тестов, а также вопросы использования API инфраструктуры тестирования и включения тестов в автоматизированную процедуру сборки. В главе 8 даются рекомендации по созданию тестов, которые были бы удобны для чтения и сопровождения и заслуживали доверия.

В четвертой части речь идет о внедрении новой методологии в организации и о работе с уже существующим кодом. В главе 9 обсуждаются проблемы, с которыми приходится сталкиваться при попытке внедрить автономное тестирование в организации, и способы их решения. Здесь же перечисляются вопросы, которые вам могут задать, и предлагаются ответы на них. Глава 10 посвящена автономному тестированию существующего унаследованного кода. Описываются два способа решить, с чего начинать тестирование, и рассматриваются некоторые инструменты тестирования нетестопригодного кода. В главе 11 мы поговорим о весьма важной теме проектирования с учетом тестопригодности и о существующих сегодня вариантах.

В приложении описываются инструменты, которые могут оказаться полезны для тестирования.

Графические выделения и загрузка исходного кода

Исходный код к этой книге можно скачать со страницы https://github.com/royosherove/aout2, с сайта книги по адресу www.ArtOfUnitTesting.com, а также с сайта издательства www.manning.