

Искусство автономного тестирования с примерами на C#



Рой Ошероув

Второе
издание

DMK
ИЗДАТЕЛЬСТВО



MANNING

*Искусство
автономного
тестирования
с примерами на C#*

Второе издание

РОЙ ОШЕРОУВ



Москва, 2016

Ошероув, Р. Искусство автономного тестирования с примерами на С# / Р. Ошероув. — 2-е изд. — Санкт-Петербург : ДМК Пресс, 2016. — 359 с. : ил. — Библиогр. : с. 351—359.

УДК 004.415.53.031.2

ББК 32

Чит. зал №1 — 2 экз.

Во втором издании книги «Искусство автономного тестирования» автор шаг за шагом проведет вас по пути от первого простенького автономного теста до создания полного комплекта тестов - понятных, удобных для сопровождения и заслуживающих доверия. Вы и не заметите, как перейдете к более сложным вопросам - заглушкам и подставкам - и попутно научитесь работать с изолирующими каркасами типа Moq, FakeItEasy или Туреmock Isolator. Вы узнаете о паттернах тестирования и организации тестов, о том, как проводить рефакторинг приложений и тестировать «нетестопригодный» код. Не забыл автор и об интеграционном тестировании и тестировании работы с базами данных.

Примеры в книге написаны на С#, но будут понятны всем, кто владеет каким-нибудь статически типизированным языком, например Java или С++.



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. Текущий 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. Смещение 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 тестами и 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. Именованые автономных тестов	256
8.3.2. Именованые 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 Turpmock 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. Моq.....	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. Сарубара	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 CHES	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