Оглавление

Предисловие к русскому изданию **14**

Предисловие к оригинальному изданию **15**

Благодарности **16**

О книге **17**

Для кого написана эта книга 17

Структура книги 18

О коде 18

Форум для обсуждения книги 19

Об авторе 19

Иллюстрация на обложке 19

От издательства 20

Часть I. Общая картина 21

Глава 1. Цель юнит-тестирования **22**

1.1. Текущее состояние дел в юнит-тестировании 23

1.2. Цель юнит-тестирования 24

1.2.1. В чем разница между плохими и хорошими тестами? 26

1.3. Использование метрик покрытия для оценки качества тестов 28
1.3.1. Метрика покрытия code coverage 28
1.3.2. Branch coverage 30
1.3.3. Проблемы с метриками покрытия....31
1.3.4. Процент покрытия как цель 34
1.4. Какими должны быть успешные тесты? 35
1.4.1. Интеграция в цикл разработки....35
1.4.2. Проверка только самых важных частей кода 35
1.4.3. Максимальная защита от багов при минимальных затратах на сопровождение....36
1.5. Что вы узнаете из книги 37
Итоги....38

Глава 2. Что такое юнит-тест? 40

2.1. Определение юнит-теста 40
2.1.1. Вопрос изоляции: лондонская школа 41
2.1.2. Вопрос изоляции: классический подход....47
2.2. Классическая и лондонская школы юнит-тестирования....50
2.2.1. Работа с зависимостями в классической и лондонской школах....51
2.3. Сравнение классической и лондонской школ юнит-тестирования 54
2.3.1. Юнит-тестирование одного класса за раз....55
2.3.2. Юнит-тестирование большого графа взаимосвязанных классов....56
2.3.3. Выявление точного местонахождения ошибки 57
2.3.4. Другие различия между классической и лондонской школами 57
2.4. Интеграционные тесты в двух школах 58
2.4.1. Сквозные (end-to-end) тесты как подмножество интеграционных тестов 60
Итоги....62

Глава 3. Анатомия юнит-теста 64

3.1. Структура юнит-теста 65
3.1.1. Паттерн AAA....65
3.1.2. Избегайте множественных секций arrange, act и assert 66
3.1.3. Избегайте команд if в тестах....67
3.1.4. Насколько большой должна быть каждая секция?....68

3.1.5. Сколько проверок должна содержать секция проверки?....70
3.1.6. Нужна ли завершающая (teardown) фаза?....71
3.1.7. Выделение тестируемой системы....71
3.1.8. Удаление комментариев «arrange/act/assert» из тестов....72
3.2. Фреймворк тестирования xUnit....72
3.3. Переиспользование тестовых данных между тестами....74
3.3.1. Сильная связность (high coupling) между тестами как антипаттерн....75
3.3.2. Использование конструкторов в тестах ухудшает читаемость....76
3.3.3. Более эффективный способ переиспользования тестовых данных....76
3.4. Именование юнит-тестов....78
3.4.1. Рекомендации по именованию юнит-тестов....80
3.4.2. Пример: переименование теста в соответствии с рекомендациями....80
3.5. Параметризованные тесты....82
3.5.1. Генерирование данных для параметризованных тестов....85
3.6. Использование библиотек для дальнейшего улучшения читаемости тестов....86
Итоги....88

Часть II. Обеспечение эффективной работы ваших тестов....89

Глава 4. Четыре аспекта хороших юнит-тестов....90

4.1. Четыре аспекта хороших юнит-тестов....91
4.1.1. Первый аспект: защита от багов....91
4.1.2. Второй аспект: устойчивость к рефакторингу....92
4.1.3. Что приводит к ложным срабатываниям?....94
4.1.4. Тестирование конечного результата вместо деталей имплементации....98
4.2. Связь между первыми двумя атрибутами....99
4.2.1. Максимизация точности тестов....100
4.2.2. Важность ложных и ложноотрицательных срабатываний: динамика....101
4.3. Третий и четвертый аспекты: быстрая обратная связь и простота поддержки....103
4.4. В поисках идеального теста....103
4.4.1. Возможно ли создать идеальный тест?....104

4.4.2. Крайний случай № 1: сквозные (end-to-end) тесты ... 105
4.4.3. Крайний случай № 2: тривиальные тесты ... 106
4.4.4. Крайний случай № 3: хрупкие тесты ... 106
4.4.5. В поисках идеального теста: результаты ... 108
4.5. Известные концепции автоматизации тестирования ... 111
4.5.1. Пирамида тестирования ... 111
4.5.2. Выбор между тестированием по принципу «черного ящика» и «белого ящика» ... 114
Итоги ... 115

Глава 5. Моки и хрупкость тестов ... 117

5.1. Отличия моков от стабов ... 118
5.1.1. Разновидности тестовых заглушек ... 118
5.1.2. Мок-инструмент и мок — тестовая заглушка ... 120
5.1.3. Не проверяйте взаимодействия со стабами ... 121
5.1.4. Использование моков вместе со стабами ... 122
5.1.5. Связь моков и стабов с командами и запросами ... 123
5.2. Наблюдаемое поведение и детали имплементации ... 124
5.2.1. Наблюдаемое поведение — не то же самое, что публичный API ... 125
5.2.2. Утечка деталей имплементации: пример с операцией ... 126
5.2.3. Хорошо спроектированный API и инкапсуляция ... 129
5.2.4. Утечка деталей имплементации: пример с состоянием ... 130
5.3. Связь между моками и хрупкостью тестов ... 132
5.3.1. Определение гексагональной архитектуры ... 132
5.3.2. Внутрисистемные и межсистемные взаимодействия ... 136
5.3.3. Внутрисистемные и межсистемные взаимодействия: пример ... 138
5.4. Еще раз о различиях между классической и лондонской школами юнит-тестирования ... 141
5.4.1. Не все внепроцессные зависимости должны заменяться моками ... 142
5.4.2. Использование моков для проверки поведения ... 144
Итоги ... 144

Глава 6. Стили юнит-тестирования ... 147

6.1. Три стиля юнит-тестирования ... 148
6.1.1. Проверка выходных данных ... 148

6.1.2. Проверка состояния ... 149
6.1.3. Проверка взаимодействий ... 150
6.2. Сравнение трех стилей юнит-тестирования ... 151
6.2.1. Сравнение стилей по метрикам защиты от багов и быстроте обратной связи ... 152
6.2.2. Сравнение стилей по метрике устойчивости к рефакторингу ... 152
6.2.3. Сравнение стилей по метрике простоты поддержки ... 153
6.2.4. Сравнение стилей: результаты ... 156
6.3. Функциональная архитектура ... 157
6.3.1. Что такое функциональное программирование? ... 157
6.3.2. Что такое функциональная архитектура? ... 161
6.3.3. Сравнение функциональных и гексагональных архитектур ... 163
6.4. Переход на функциональную архитектуру и тестирование выходных данных ... 164
6.4.1. Система аудита ... 165
6.4.2. Использование моков для отделения тестов от файловой системы ... 167
6.4.3. Рефакторинг для перехода на функциональную архитектуру ... 170
6.4.4. Потенциальные будущие изменения ... 176
6.5. Недостатки функциональной архитектуры ... 177
6.5.1. Применимость функциональной архитектуры ... 177
6.5.2. Недостатки по быстродействию ... 179
6.5.3. Увеличение размера кодовой базы ... 179
Итоги ... 180

Глава 7. Рефакторинг для получения эффективных юнит-тестов ... 182

7.1. Определение кода для рефакторинга ... 183
7.1.1. Четыре типа кода ... 183
7.1.2. Использование паттерна «Простой объект» для разделения переусложненного кода ... 187
7.2. Рефакторинг для получения эффективных юнит-тестов ... 190
7.2.1. Знакомство с системой управления клиентами ... 190
7.2.2. Версия 1: преобразование неявных зависимостей в явные ... 192
7.2.3. Версия 2: уровень сервисов приложения ... 193
7.2.4. Версия 3: вынесение сложности из сервисов приложения ... 195
7.2.5. Версия 4: новый класс Company ... 197

7.3. Анализ оптимального покрытия юнит-тестов ... 200
7.3.1. Тестирование слоя предметной области и вспомогательного кода ... 200
7.3.2. Тестирование кода из трех других четвертей ... 201
7.3.3. Нужно ли тестировать предусловия? ... 202
7.4. Условная логика в контроллерах ... 203
7.4.1. Паттерн «CanExecute/Execute» ... 205
7.4.2. Использование доменных событий для отслеживания изменений доменной модели ... 208
7.5. Заключение ... 212
Итоги ... 214

Часть III. Интеграционное тестирование ... 217

Глава 8. Для чего нужно интеграционное тестирование? ... 218

8.1. Что такое интеграционный тест? ... 219
8.1.1. Роль интеграционных тестов ... 219
8.1.2. Снова о пирамиде тестирования ... 220
8.1.3. Интеграционное тестирование и принцип Fail Fast ... 222
8.2. Какие из внепроцессных зависимостей должны проверяться напрямую ... 224
8.2.1. Два типа внепроцессных зависимостей ... 224
8.2.2. Работа с управляемыми и неуправляемыми зависимостями ... 225
8.2.3. Что делать, если вы не можете использовать реальную базу данных в интеграционных тестах? ... 226
8.3. Интеграционное тестирование: пример ... 227
8.3.1. Какие сценарии тестировать? ... 228
8.3.2. Классификация базы данных и шины сообщений ... 229
8.3.3. Как насчет сквозного тестирования? ... 229
8.3.4. Интеграционное тестирование: первая версия ... 231
8.4. Использование интерфейсов для абстрагирования зависимостей ... 232
8.4.1. Интерфейсы и слабая связность ... 232
8.4.2. Зачем использовать интерфейсы для внепроцессных зависимостей? ... 233
8.4.3. Использование интерфейсов для внутрипроцессных зависимостей ... 235
8.5. Основные приемы интеграционного тестирования ... 235

8.5.1. Явное определение границ модели предметной области ... 235
8.5.2. Сокращение количества слоев ... 236
8.5.3. Исключение циклических зависимостей ... 237
8.5.4. Использование нескольких секций действий в тестах ... 240
8.6. Тестирование функциональности логирования ... 241
8.6.1. Нужно ли тестировать функциональность логирования? ... 241
8.6.2. Как тестировать функциональность логирования? ... 243
8.6.3. Какой объем логирования можно считать достаточным? ... 248
8.6.4. Как передавать экземпляры логеров? ... 249
8.7. Заключение ... 250
Итоги ... 250

Глава 9. Рекомендации при работе с моками ... 254

9.1. Достижение максимальной эффективности моков ... 254
9.1.1. Проверка взаимодействий на границах системы ... 257
9.1.2. Замена моков шпионами ... 261
9.1.3. Как насчет IDomainLogger? ... 263
9.2. Практики мокирования ... 263
9.2.1. Моки только для интеграционных тестов ... 264
9.2.2. Несколько моков на тест ... 264
9.2.3. Проверка количества вызовов ... 265
9.2.4. Используйте моки только для принадлежащих вам типов ... 265
Итоги ... 267

Глава 10. Тестирование базы данных ... 268

10.1. Предусловия для тестирования базы данных ... 269
10.1.1. Хранение базы данных в системе контроля версий ... 269
10.1.2. Справочные данные являются частью схемы базы данных ... 270
10.1.3. Отдельный экземпляр для каждого разработчика ... 271
10.1.4. Развертывание базы данных на основе состояния и на основе миграций ... 271
10.2. Управление транзакциями ... 274
10.2.1. Управление транзакциями в рабочем коде ... 274
10.2.2. Управление транзакциями в интеграционных тестах ... 282

10.3. Жизненный цикл тестовых данных ... 284
10.3.1. Параллельное или последовательное выполнение тестов? ... 284
10.3.2. Очистка данных между запусками тестов ... 285
10.3.3. Не используйте базы данных в памяти ... 287
10.4. Переиспользование кода в секциях тестов ... 287
10.4.1. Переиспользование кода в секциях подготовки ... 288
10.4.2. Переиспользование кода в секциях действий ... 290
10.4.3. Переиспользование кода в секциях проверки ... 291
10.4.4. Не создает ли тест слишком много транзакций? ... 292
10.5. Типичные вопросы при тестировании баз данных ... 293
10.5.1. Нужно ли тестировать операции чтения? ... 294
10.5.2. Нужно ли тестировать репозитории? ... 294
10.6. Заключение ... 296
Итоги ... 297

Часть IV. Антипаттерны юнит-тестирования ... 299

Глава 11. Антипаттерны юнит-тестирования ... 300

11.1. Юнит-тестирование приватных методов ... 300
11.1.1. Приватные методы и хрупкость тестов ... 301
11.1.2. Приватные методы и недостаточное покрытие ... 301
11.1.3. Когда тестирование приватных методов допустимо ... 302
11.2. Раскрытие приватного состояния ... 304
11.3. Утечка доменных знаний в тесты ... 306
11.4. Загрязнение кода ... 307
11.5. Мокирование конкретных классов ... 310
11.6. Работа со временем ... 313
11.6.1. Время как неявный контекст ... 313
11.6.2. Время как явная зависимость ... 314
11.7. Заключение ... 315
Итоги ... 315

Посвящаю моей жене Нине

Предисловие к русскому изданию

Я помню, как начинал работать программистом в 2004 году в небольшой московской компании. В те времена никто не только не писал юнит-тесты, многие даже не знали о такой практике. Сейчас юнит-тестирование — неотъемлемая часть любого сколько-нибудь крупного проекта. Причем неважно, работаете ли вы в России или трудитесь на аутсорсе, навык написания хороших, легких в сопровождении тестов необходим всем.

Несмотря на такую востребованность, найти информацию о том, как именно писать такие юнит-тесты, непросто. Существует множество туториалов, где показывают, как дать на вход калькулятору два числа и проверить возвращаемое значение или как использовать мок-библиотеку, но это по сути все. Писать эффективные тесты часто приходится, учась на своих ошибках.

Цель этой книги — не учить пользоваться фреймворками юнит-тестирования или настраивать TeamCity для регулярного прогона тестов, этого материала в интернете предостаточно. Ее цель — собрать воедино всю информацию о написании эффективных, простых в поддержке тестов. Большую часть этой информации можно узнать лишь опытным способом, набив по пути много шишек. Эта книга позволит вам пропустить стадию шишек и перейти сразу к плодам этих проб и ошибок, на что у меня ушло около 10 лет.

Предисловие к оригинальному изданию

Помню свой первый проект, в котором применил юнит-тестирование. Все прошло относительно хорошо, но после того как он был закончен, я взглянул на тесты и подумал, что многие из них были напрасной тратой времени. Большинство моих юнит-тестов тратило изрядную долю времени на настройку ожиданий и плетение сложной паутины зависимостей — и все это для проверки правильности всего трех строк кода в моем контроллере. Я не мог сформулировать, что именно не так с моими тестами, но было стойкое ощущение того, что это не нормально. К счастью, я не отказался от юнит-тестирования и продолжал применять его в последующих проектах. Тем не менее я чувствовал все большее и большее несогласие с общепринятыми (на тот момент) практиками юнит-тестирования. За эти годы я часто писал о юнит-тестировании. В этих статьях мне наконец удалось сформулировать, что пошло не так с моими первыми тестами, и обобщить эти знания для более широких областей юнит-тестирования. Эта книга — результат моих исследований, проб и ошибок, тщательно переработанных, уточненных и собранных в одном месте.

Я получил математическое образование и твердо считаю, что рекомендации в программировании, как и теоремы в математике, должны выводиться из фундаментальных принципов. Я постарался структурировать эту книгу аналогичным образом: не делать поспешных выводов и не выступать с необоснованными заявлениями, а начать с чистого листа: установить фундаментальные принципы и выводить все рекомендации по юнит-тестированию из этих принципов, «с нуля». Интересно, что после установления аксиоматики рекомендации часто выводятся сами собой, как простые следствия.

Юнит-тестирование постепенно становится обязательным требованием для программных проектов, и эта книга даст вам все необходимое для построения хороших и простых в сопровождении тестов.

Благодарности

На написание книги ушло много времени. Хотя я и был к этому готов, работы все равно оказалось намного больше, чем я мог себе представить.

Хочу поблагодарить многих людей: Сэма Зейдела (Sam Zaydel), Алессандро Кампейса (Alessandro Campeis), Фрэнсис Буран (Frances Buran), Тиффани Тейлор (Tiffany Taylor) и особенно Марину Майклз (Marina Michaels), чье бесценное мнение помогло мне поддерживать качество материала на высшем уровне и попутно улучшило мои писательские навыки. Также спасибо остальным сотрудникам Manning, работавшим над книгой в процессе выпуска и оставшимся незамеченными.

Также хочу поблагодарить научных редакторов, которые не пожалели времени на чтение моей рукописи в различных фазах работы и предоставили полезнейшую обратную связь: Аарона Бартона (Aaron Barton), Алессандро Кампейса (Alessandro Campeis), Конора Редмонда (Conor Redmond), Дрор Хелпер (Dror Helper), Грега Райта (Greg Wright), Хемант Конеру (Hemant Koneru), Джереми Ланге (Jeremy Lange), Хорхе Эзекиля Бо (Jorge Ezequiel Bo), Джорта Роденбурга (Jort Rodenburg), Марка Ненадова (Mark Nenadov), Марко Умека (Marko Umek), Маркуса Мецкера (Markus Matzker), Шрихари Шридхарана (Srihari Sridharan), Стивена Джона Уорнетта (Stephen John Warnett), Суманта Тамбе (Sumant Tambe), Тима ван Дьюрзена (Tim van Deurzen) и Владимира Купцова (Vladimir Kuptsov).

Но больше всего хочу поблагодарить свою жену Нину, которая поддерживала меня на протяжении всей работы над книгой.

О книге

В книге «Принципы юнит-тестирования» подробно рассматриваются рекомендации, паттерны и антипаттерны, встречающиеся в области юнит-тестирования. После чтения этой книги вы будете знать все необходимое для того, чтобы стать экспертом в области создания успешных проектов — проектов, которые легко расширять и сопровождать благодаря хорошим тестам.

Для кого написана эта книга

У большинства сетевых и печатных ресурсов имеется один недостаток: они подробно излагают основы юнит-тестирования, но практически не выходят за эти рамки. Такие ресурсы могут быть очень ценными, однако обучение на этом не заканчивается. Существует и следующий уровень: умение не просто писать тесты, но делать это так, чтобы ваши усилия приносили максимальную отдачу. К сожалению, многим людям приходится самим разбираться, как выйти на этот уровень, часто методом проб и ошибок. Эта книга поможет вам в этом. В ней приводится точное определение того, что собой представляет качественный тест. Это определение формирует единую систему отсчета, которая поможет вам взглянуть на многие из ваших тестов в новом свете и увидеть, какие из них работают на пользу проекта, а какие следует отрефакторить или вообще удалить.

Если у вас мало опыта в юнит-тестировании, из этой книги вы многое узнаете. Опытный программист, скорее всего, уже понимает некоторые идеи, изложенные здесь. Книга поможет ему осознать, почему приемы и практики, которыми он пользовался все это время, настолько полезны. И не стоит недооценивать этот навык: умение четко донести свои идеи коллегам чрезвычайно полезно.

Структура книги

Одиннадцать глав этой книги разделены на четыре части. В части I изложены основы юнит-тестирования, а также напоминаются наиболее общие практики юнит-тестирования:

- Глава 1 показывает цели юнит-тестирования, в ней приводится краткий обзор того, как отличить хороший тест от плохого.
- В главе 2 анализируется определение юнит-тестирования и обсуждаются две основные школы в области юнит-тестирования.
- Глава 3 рассматривает некоторые базовые вопросы — такие как структура юнит-тестов, переиспользование тестовых данных и параметризация тестов.

В части II мы перейдем к сути дела — вы увидите, какими свойствами должен обладать хороший юнит-тест, а также узнаете, как провести рефакторинг тестов для повышения их качества:

- В главе 4 определяются четыре характеристики, по которым можно оценить качество теста, а также предоставляется общая система координат, которая используется на протяжении всей книги.
- В главе 5 объясняется, для чего нужны моки (mocks), и анализируется их связь с хрупкостью тестов.
- В главе 6 рассматриваются три стиля юнит-тестирования и то, какой из этих стилей производит тесты лучшего качества и почему.
- Глава 7 показывает, как провести рефакторинг раздутых, чрезмерно усложненных тестов и получить тесты, сочетающие в себе максимальную эффективность с минимальными затратами на сопровождение.

В части III изучаются вопросы интеграционного тестирования:

- В главе 8 рассматривается интеграционное тестирование в целом, его достоинства и недостатки.
- В главе 9 обсуждаются моки (mocks) и как работать с ними так, чтобы максимально повысить эффективность ваших тестов.
- В главе 10 рассматривается работа с реляционными базами данных в тестах.

В главе 11 части IV представлены стандартные антипаттерны юнит-тестирования.

О коде

Примеры кода написаны на C#, но те аспекты, которые они демонстрируют, применимы к любому объектно-ориентированному языку (например, Java или C++). Я старался не пользоваться специфическими языковыми возможностями C#

и сделать код примеров по возможности простым, чтобы вы легко разобрались в нем. Весь код примеров можно скачать по адресу www.manning.com/books/unit-testing.

Форум для обсуждения книги

Приобретая книгу, вы получаете бесплатный доступ к закрытому веб-форуму Manning, на котором можно публиковать комментарии по поводу книги, задавать технические вопросы и получать помощь от автора и других пользователей. Чтобы получить доступ к форуму, откройте страницу https://livebook.manning.com/#!/book/unit-testing/discussion. За информацией о форумах Manning и правилах поведения обращайтесь по адресу https://livebook.manning.com/#!/discussion.

В рамках своих обязательств перед читателями издательством Manning предоставляет ресурс для проведения осмысленного диалога между читателями и автором. Эти обязательства не подразумевают конкретный вклад со стороны автора, участие которого в работе форума остается добровольным (и неоплачиваемым). Задавайте автору интересные вопросы, чтобы он не терял интереса к происходящему! Форум и архивы предшествующих обсуждений доступны на веб-сайте издателя, пока книга находится в печати.

Другие сетевые ресурсы

- Мой блог находится по адресу EnterpriseCraftsmanship.com.
- У меня также имеется онлайн-курс по юнит-тестированию (в данный момент находится в разработке), на который можно записаться по адресу UnitTestingCourse.com.

Об авторе

Владимир Хориков — разработчик, Microsoft MVP и автор на платформе Pluralsight. Профессионально занимается разработкой программного обеспечения более 15 лет, а также обучением команд тонкостям юнит-тестирования. За последние годы Владимир опубликовал несколько популярных серий в блогах, а также онлайн-курс на тему юнит-тестирования. Главное достоинство его стиля обучения, которое часто отмечают студенты, — сильная теоретическая подготовка, которая затем используется на практике.

Иллюстрация на обложке

Иллюстрация, помещенная на обложку второго издания книги и озаглавленная «Esthinienne», была позаимствована из изданного в 1788 г. каталога национальных

костюмов Жака Грассе де Сен-Совера (1757–1810) «Costumes Civils Actuels de Tous les Peuples Connus». Каждая иллюстрация нарисована и раскрашена от руки. Иллюстрации из каталога Грассе де Сен-Совера напоминают о культурных различиях между городами и весями мира, имевших место почти двести лет назад. Люди, проживавшие в изолированных друг от друга регионах, говорили на разных языках и диалектах. По одежде человека можно было определить, в каком городе, поселке или поселении он проживает.

С тех пор дресс-код сильно изменился, да и различия между разными регионами стали не столь выраженными. В наше время довольно трудно узнать жителей разных континентов, не говоря уже о жителях разных городов или регионов. Возможно, мы отказались от культурных различий в пользу более разнообразной личной жизни — и конечно, в пользу более разнообразной и стремительной технологической жизни.

Сейчас, когда все компьютерные книги похожи друг на друга, издательство Manning стремится к разнообразию и помещает на обложки книг иллюстрации, показывающие особенности жизни в разных странах мира два века назад.

От издательства

Ваши замечания, предложения, вопросы отправляйте по адресу comp@piter.com (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

На веб-сайте издательства www.piter.com вы найдете подробную информацию о наших книгах.

Часть I

Общая картина

Эта часть книги вводит читателя в курс текущего состояния дел в области юнит-тестирования. В главе 1 я расскажу о цели юнит-тестирования и покажу, как отличить хороший тест от плохого. Мы поговорим о метриках тестового покрытия и обсудим свойства хорошего юнит-теста.

В главе 2 будет приведено определение юнит-теста. Незначительное на первый взгляд расхождение в этом определении привело к формированию двух школ юнит-тестирования, которые будут описаны в этой главе. Глава 3 — это памятка по некоторым базовым темам, таким как структура юнит-тестов, переиспользование тестовых данных и параметризация тестов.

1 Цель юнит-тестирования

В этой главе:

- ✓ Состояние дел в юнит-тестировании.
- ✓ Цель юнит-тестирования.
- ✓ Последствия от написания плохих тестов.
- ✓ Использование метрик тестового покрытия для оценки качества тестов.
- ✓ Атрибуты успешных тестов.

Изучение юнит-тестирования не заканчивается на освоении его технических сторон: тестового фреймворка, библиотеки моков и т. д. Юнит-тестирование не сводится к простому написанию тестов. Важно стремиться к тому, чтобы свести к минимуму усилия, потраченные на написание тестов, и максимизировать преимущества, которые они приносят. Совместить эти две задачи не так просто.

Наблюдать за проектами, добившимися заветного баланса, одно удовольствие: они развиваются без лишних усилий, не требуют особого сопровождения и быстро адаптируются к постоянно изменяющимся потребностям заказчиков. С другой стороны, наблюдать за проектами, которые не справились с этой задачей, крайне мучительно. Несмотря на все усилия и впечатляющее количество юнит-тестов, такие проекты развиваются медленно, содержат множество багов и требуют больших затрат на сопровождение.

Существуют различные методы юнит-тестирования. Одни дают отличные результаты и помогают поддерживать качество кода на должном уровне. Другие с этим не справляются: полученные тесты не приносят особой пользы, часто ломаются и требуют значительных усилий при сопровождении.

Эта книга поможет вам отличать плохие методы юнит-тестирования от хороших. Вы узнаете, как анализировать эффективность ваших тестов и применить подходящие методы тестирования в вашей конкретной ситуации. Также вы научитесь обходить распространенные антипаттерны — паттерны, который на первый взгляд выглядят разумно, но приводят к проблемам в будущем.

Но начнем с азов. В этой главе приводится краткий обзор состояния дел в юнит-тестировании, описывается цель написания тестов, а также дается представление о том, что собой представляют успешные тесты.

1.1. Текущее состояние дел в юнит-тестировании

За два последних десятилетия программная индустрия начала постепенно практиковать юнит-тестирование. Во многих компаниях эти практики уже считаются обязательными — многие программисты пишут юнит-тесты и понимают их важность. Разногласий относительно того, нужно ли заниматься юнит-тестированием, уже нет.

При разработке корпоративных приложений практически каждый проект включает какое-то количество юнит-тестов. Соотношение между рабочим и тестовым кодом обычно лежит в диапазоне от 1:1 до 1:3 (на каждую строку рабочего кода приходится от одной до трех строк тестового кода). Иногда это соотношение достигает существенно большего значения — вплоть до 1:10.

Но как и все новые технологии, юнит-тестирование продолжает развиваться. С вопроса «нужно ли писать юнит тесты?» обсуждение перешло в другую плоскость: как писать хорошие юнит-тесты? Именно в этой области кроются основные разногласия.

Результаты этих разногласий проявляются в программных проектах. Многие проекты содержат автоматизированные тесты, но они зачастую не приносят результатов, на которые надеются разработчики: сопровождение проектов и разработка в них нового функционала все так же требуют значительных усилий, а в уже написанном функционале постоянно появляются новые ошибки. Юнит-тесты, которые вроде бы должны помогать, никак не способствуют решению этих проблем. Иногда они даже усугубляют ситуацию.

Это плачевная ситуация, и она часто возникает из-за того, что юнит-тесты не справляются со своей задачей. Различия между хорошими и плохими тестами не ограничиваются вкусами или личными предпочтениями. На практике эти различия влияют на весь проект — они могут либо помочь вам успешно завершить проект, либо привести к его провалу.

Важно понимать, какими качествами должен обладать хороший юнит-тест. И тем не менее информацию на эту тему найти довольно сложно. В интернете существуют

разрозненные статьи и выступления с конференций, но я еще не видел ни одного исчерпывающего материала по этой теме.

Ситуация с книгами ненамного лучше; многие из них сосредоточены на основах юнит-тестирования, но не выходят за эти рамки. Конечно, такие книги тоже полезны, особенно если вы только начинаете осваивать юнит-тестирование. Но обучение не заканчивается на основах. Важно не просто писать тесты, но делать это так, чтобы усилия приносили максимальную отдачу.

ЧТО ТАКОЕ «КОРПОРАТИВНОЕ ПРИЛОЖЕНИЕ»?

Корпоративным (enterprise) называется приложение, предназначенное для автоматизации внутренних процессов компании. Существует много разновидностей корпоративных приложений, но обычно оно обладает следующими характеристиками:

- ✓ высокая сложность бизнес-логики;
- ✓ большой срок жизни проекта;
- ✓ умеренные объемы данных;
- ✓ низкие или средние требования к быстродействию.

Эта книга поможет вам в этом. В ней приводится точное, исчерпывающее определение качественного юнит-теста. Вы увидите, как это определение применяется к практическим примерам из реальной жизни.

Книга принесет наибольшую пользу, если вы занимаетесь разработкой корпоративных приложений, но основные идеи применимы в любом программном проекте.

1.2. Цель юнит-тестирования

Прежде чем углубляться в тему юнит-тестирования, давайте рассмотрим, для чего вообще нужно юнит-тестирование и какой цели оно помогает добиться. Считается, что юнит-тестирование улучшает качество кода проекта. И это правда: необходимость написания юнит-тестов обычно приводит к улучшению качества кода. Но это не главная цель юнит-тестирования, а всего лишь приятный побочный эффект.

Какова же тогда цель юнит-тестирования? Его цель — обеспечение *стабильного* роста программного проекта. Ключевым словом здесь является «стабильный». В начале жизни проекта развивать его довольно просто. Намного сложнее поддерживать это развитие с прошествием времени.

На рис. 1.1 изображена динамика роста типичного проекта без тестов. Все начинается быстро, потому что ничего вас не тормозит. Еще не приняты неудачные архитектурные решения; еще нет существующего кода, который необходимо прорабатывать и поддерживать. Однако с течением времени вам приходится тратить все больше времени, чтобы написать тот же по объему функционал, что и в начале