Введение

Как уже было сказано, разработка программного обеспечения без дефектов — исключительно сложный вид деятельности. Доказательство корректности реальных систем до сих пор остается за пределами возможностей, а описание поведения оказывается настолько же сложным. Любое предсказание будущего может оказаться ложным. Если бы этот навык давал гарантированный результат, мы стали бы игроками на бирже, а не разработчиками.

Автоматизированная проверка корректности поведения программного обеспечения является одним из важнейших улучшений методов разработки за последние несколько десятков лет. Данная практика оказала значительное влияние на повышение производительности разработчиков, повышение качества и защиту программного обеспечения от сбоев. Сам факт, что разработчики применяют эту методику по собственной воле, говорит об эффективности технологии.

Во введении описывается концепция автоматизации тестов с помощью ряда инструментов (включая xUnit), перечисляются причины применения этой методики и демонстрируются сложности на пути к правильной автоматизации тестирования.

Обратная связь

Обратная связь является важным элементом многих видов деятельности. Она позволяет убедиться, что выполняемые действия дают желаемый эффект. Чем быстрее работает обратная связь, тем быстрее реакция. Хорошим примером такой обратной связи являются дорожные отбойники между полотном и обочиной многих шоссе. Если съехать на обочину, сразу понятно, что колеса находятся не на дороге. Но, получив эту информацию раньше (когда колесо выходит на отбойник), можно изменить направление движения и снизить вероятность съезда с дороги.

Тестирование обеспечивает обратную связь при разработке программного обеспечения. Такая связь является неотъемлемым элементом “гибкого” процесса разработки. Циклы обратной связи позволяют сохранить уверенность в создаваемом программном обеспечении, ускорить разработку и уменьшить ненужные переживания о качестве. Тесты предоставляют возможность спокойно добавлять новую функциональность, показывая состояние старой.

Тестирование

Традиционное определение “тестирования” происходит из мира контроля качества. Тестировать программное обеспечение необходимо, так как в нем содержатся ошибки!

Поэтому ПО тестируется, тестируется, тестируется и тестируется еще, до тех пор пока становится невозможно доказать существование ошибок в коде. Традиционно данный процесс происходит уже после завершения разработки. В результате он становится методом контроля, а не обеспечения качества продукта. Во многих организациях тестирование выполняется не разработчиками. Обратная связь с таким процессом является очень ценной, но на поздних этапах разработки эта ценность значительно снижается. Кроме того, неприятным эффектом такой организации процесса является увеличение цикла разработки, так как найденные проблемы передаются разработчикам для исправления, требуя повторного цикла тестирования. Возникает вопрос: "Какая методика тестирования позволит разработчикам получать более быструю обратную связь?"

Тестирование разработчиками

Редкий разработчик верит, что может сразу написать полностью работающий код. На самом деле многие приятно удивляются, если что-то заработает сразу. (Надеюсь, ничьи иллюзии не были разбиты жестокой реальностью?)

Поэтому разработчики также занимаются тестированием. В процессе написания кода необходимо доказательство корректности работы программного обеспечения. Некоторые разработчики тестируют тем же способом, что и тестеры, рассматривая целую систему как единую сущность. Но большинство предпочитают модульное тестирование. "Модули" могут быть большими компонентами или отдельными классами, методами или функциями. Ключевым отличием этих тестов является выбор модулей в соответствии с дизайном программного продукта, а не на основе прямой трансляции требований. (Некоторая часть модульных тестов может непосредственно соответствовать бизнес-логике или пользовательским тестам, но большая часть тестов связана с кодом, окружающим бизнес-логику.)

Автоматизированное тестирование

Автоматизированное тестирование существует уже несколько десятков лет. Работая в начале 1980-х годов над системой коммутации телефонных звонков в компании Bell-Northern Researsh, которая вела разработки для компании Nortel, автору приходилось заниматься автоматизированным регрессионным и стресс-тестированием создаваемых программных и аппаратных компонентов. Тестирование выполнялось в контексте "системного теста" с помощью аппаратных и программных средств, которые программировались с помощью тестовых сценариев. Тестовые устройства подключались к коммутатору и играли роль нескольких телефонов и телефонных коммутаторов, генерируя телефонные звонки и запрашивая другие функции телефонии. Конечно, такая инфраструктура автоматизированного тестирования не была предназначена для модульного тестирования, а также была недоступна разработчикам из-за огромного количества аппаратных компонентов.

За последние десять лет появились более универсальные инструменты автоматизации тестирования, позволяющие взаимодействовать с приложениями через пользовательский интерфейс. Некоторые из них применяют языки сценариев для определения тестов. Более развитые продукты основаны на метафоре "робота" или "записи и воспроизведения" для автоматизации тестирования. К сожалению, опыт использования таких инструмен-

тов оказался не очень положительным. Причиной этому стала повышенная стоимость обслуживания тестов, связанная с проблемой "теста хрупкого".

Проблема "хрупкого теста"

Автоматизация тестирования на основе коммерческих приложений с "записью и воспроизведением" или "роботом" заработала не очень хорошую репутацию среди пользователей. Такие тесты часто завершались неудачно по, на первый взгляд, тривиальным причинам. Важно понимать ограничения такого способа автоматизации тестов, чтобы не наткнуться на известные "подводные камни" — чувствительность к поведению, чувствительность к интерфейсу, чувствительность к данным и чувствительность к контексту.

Чувствительность к поведению

Если поведение системы изменилось (например, если изменились требования и система была модифицирована), любой тест модифицированной функциональности завершится неудачно при повторном воспроизведении (изменение в поведении может быть связано с тем, что система решает совсем другую задачу или ту же задачу, но в другой последовательности и с другими задержками). Это обычная реальность тестирования вне зависимости от подхода к автоматизации. Реальная проблема заключается в необходимости использования этой функциональности для перевода системы в подходящее для начала теста состояние. Следовательно, изменения в поведении имеют намного большее влияние на процесс тестирования, чем может показаться.

Чувствительность к интерфейсу

Нежелательно тестировать бизнес-логику системы через пользовательский интерфейс. Минимальные модификации интерфейса приведут к неудачному завершению теста, даже если человеку тест может показаться успешным. Такая неожиданная чувствительность к интерфейсу оказалась одной из причин отрицательного отношения к инструментам автоматизации в последнее десятилетие. Хотя такая проблема возникает вне зависимости от используемой технологии интерфейса, с некоторыми технологиями она усугубляется. Наиболее сложными с точки зрения взаимодействия с бизнес-логикой внутри системы являются графические интерфейсы пользователя. Наблюдаемое в последнее время смещение в сторону Web-интерфейса упростило некоторые аспекты автоматизации, но создало еще одну проблему, связанную с выполняемым кодом внутри кода HTML, который используется для более сложного взаимодействия с пользователем.

Чувствительность к данным

Каждый тест предполагает существование отправной точки, которая называется **тестовая конфигурация** (test fixture). Такой **контекст теста** (test context) иногда называется предусловием или предварительной картиной теста. Чаще всего он определяется в терминах данных, которые уже присутствуют в системе. Если данные меняются, тест может завершиться неудачно. Для решения этой проблемы могут потребоваться дополнительные усилия по обеспечению нечувствительности теста к данным.

Чувствительность к контексту

На поведение системы может оказывать влияние состояние компонентов за пределами системы. Среди таких внешних факторов можно перечислить состояние устройств (например, принтеров или серверов), других приложений и даже системных часов (например, время и/или дата запуска теста могут отличаться). Любой зависящий от контекста тест нельзя будет гарантированно повторить без получения контроля над контекстом.

Снижение чувствительности

Перечисленные типы чувствительности существуют вне зависимости от применяемой технологии автоматизации тестов. Конечно, одни технологии позволяют обойти эту проблему, а другие навязывают невыгодный способ решения. Инфраструктура автоматизации xUnit обеспечивает большую степень контроля и при необходимой квалификации может использоваться с достаточной эффективностью.

Использование автоматизированных тестов

В данной книге основное внимание уделяется регрессионному тестированию приложений. Это очень полезный тип обратной связи в процессе модификации существующего приложения, так как он позволяет обнаруживать случайно внесенные ошибки.

Тест как спецификация

Разработка на основе тестов (test driven development — TDD), являющаяся одной из ключевых практик таких гибких методов разработки, как, например, экстремальное программирование, включает в себя совершенно другое применение автоматизации тестов. В этом случае тесты используются, как спецификация поведения еще ненаписанного программного обеспечения. Эффективность разработки на основе тестов основана на разделении процесса разработки на две фазы: определение, что должна делать программа, и реализация задуманной функциональности.

Но разве сторонники гибких методов разработки не избегают каскадного процесса разработки? Да, конечно, избегают. Они предпочитают проектировать и создавать систему функция за функцией с получением работающей системы на каждом этапе с доказательством работоспособности каждой функции перед переходом к реализации следующей. Это не значит, что проектирование отсутствует; просто оно происходит непрерывно! Следование такому подходу приводит к “кристаллизации дизайна” практически без предварительного проектирования. Это не единственный возможный метод разработки. Ничто не мешает комбинировать проектирование на высоком уровне (для создания архитектуры) с дизайном на уровне отдельных функций. В любом случае желательно отложить обдумывание реализации поведения конкретного класса или метода и описать это поведение в форме выполняемой спецификации. В конце концов, многим тяжело сконцентрироваться даже на чем-то одном, так что тем более не получится концентрироваться на нескольких задачах сразу.

После написания тестов и проверки ожидаемого неудачного результата можно переходить к реализации функций, обеспечивающих успешное завершение тестов. В результате тесты позволяют оценивать оставшийся объем работ. При последовательной реали-

зации функциональности тесты один за другим будут успешно завершаться. В процессе работы успешно завершенные тесты запускаются еще раз для регрессионного тестирования, подтверждая, что внесенные изменения не имели нежелательного эффекта. В этом и заключается истинная ценность автоматизированного модульного тестирования: в возможности "фиксировать" функциональность тестируемой системы, контролируя ее неизменность.

Разработка на основе тестов

В последнее время появилось много книг по данной теме, поэтому здесь разработке на основе тестов уделяется не очень много внимания. В данной книге основное внимание уделяется коду существующих тестов, а не подходам к их написанию. Ближе всего к разработке тестов будут разделы глав книги, посвященные **рефакторингу** (refactoring) тестов, при котором тесты на основе одного шаблона превращаются в тесты на основе шаблона с другими характеристиками.

В настоящей книге предпринята попытка отделить обсуждение от конкретного процесса разработки, так как любая группа разработчиков может применять автоматизированное тестирование как в начале разработки, так и по ее завершении. Кроме того, как только разработчики научатся автоматизировать тесты уже готового кода, они с большей вероятностью предпочтут экспериментировать с созданием тестов до написания тестируемого кода. Некоторые части процесса разработки будут рассмотрены в контексте упрощения автоматизации тестирования. Относительно процессов разработки нас интересуют два аспекта.

1. Взаимодействие *полностью автоматизированных тестов* (Fully Automated Tests, с. 81), инструментария и процесса интеграции при разработке.
2. Влияние процесса разработки на простоту тестирования.

Шаблоны

При подготовке этой книги применялось множество отчетов с конференций и книг по автоматизации тестов с помощью пакета xUnit. Не удивительно, что у каждого автора есть определенная область интересов и часто применяемые техники. Хотя не со всеми описанными подходами можно согласиться, всегда стоит разобраться, почему другие авторы применяют конкретные техники и когда их решения могут оказаться более оправданными.

Такой уровень представления материала отличает его от простых примеров использования методик и шаблонов. Шаблоны помогают понять, почему применяется данная методика, позволяя принять взвешенное решение о выборе одного из альтернативных шаблонов. Правильное решение позволяет обойти нежелательные последствия в будущем.

Шаблоны в программном обеспечении существуют уже около десяти лет, поэтому большинство читателей как минимум знакомы с самим понятием. Шаблон является решением повторяющейся проблемы. Одни проблемы серьезнее других, а значит, слишком велики для решения с помощью одного шаблона. В такой ситуации может пригодиться язык шаблонов. Набор (или грамматика) шаблонов описывает пошаговый переход от общей постановки проблемы к подробному решению. В языке шаблонов одни из шабло-

нов находятся на более высоком уровне абстракции, в то время как другие сконцентрированы на проблемах более низкого уровня. Между шаблонами должны существовать связи, чтобы можно было осуществлять переход от высокоуровневых стратегий к более детальным шаблонам проектирования и далее к еще более подробным идиомам кода.

Шаблоны, принципы и запахи

В этой книге рассматриваются три типа шаблонов. Наиболее традиционным типом является повторяющееся решение для распространенной проблемы. Большинство шаблонов из этой книги попадают в данную категорию, но и они делятся на три группы.

- **Стратегии**. Это шаблоны, применение которых имеет далеко идущие последствия. Решение об использовании *общей тестовой конфигурации* (Shared Fixture, с. 350) вместо *новой тестовой конфигурации* (Fresh Fixture, с. 344) приводит к применению совершенно другого набора шаблонов тестов. Каждому стратегическому шаблону посвящен отдельный раздел в главе 6, "Стратегия автоматизации тестирования".
- **Шаблоны проектирования**. Используются при разработке тестов конкретной функциональности. Основное внимание уделяется организации логики теста. Большинству читателей будет понятен в качестве примера шаблон *подставной объект* (Mock Object, с. 558). Каждому шаблону проектирования посвящен отдельный раздел в соответствующей главе.
- **Идиомы кода**. Описывают разные способы кодирования конкретного теста. Многие идиомы имеют смысл только в пределах конкретного языка. В качестве примера можно привести блочную конструкцию (block closure) в языке Smalltalk при использовании шаблона *тест на ожидаемое исключение* (Expected Exception Test) и анонимные внутренние классы при реализации шаблона *подставной объект* (Mock Object) на языке Java. Некоторые идиомы, например *простой тест успешности* (Simple Success Test), являются настолько универсальными, что имеют соответствующие аналоги в каждом языке программирования. Обычно такие идиомы рассматриваются как варианты реализации или примеры в описании шаблона проектирования.

Часто на каждом уровне может использоваться несколько альтернативных шаблонов. Конечно, всегда существует более предпочтительный вариант, но антишаблон для одного может быть лучшим решением для другого. Поэтому в данную книгу включены шаблоны, которые не всегда рекомендуются к использованию. Здесь приводятся положительные и отрицательные стороны каждого шаблона, что позволяет принять осознанное решение об их применении. В большинстве случаев ссылка на альтернативы приводится в описании шаблона.

Выбор шаблона зависит от цели, которая должна быть достигнута в результате автоматизации теста. Такие цели поддерживаются рядом принципов, которые описывают систему признаков "хороших" автоматизированных тестов. В данной книге цели автоматизации тестирования рассматриваются в главе 3, "Цели автоматизации", а принципы описываются в главе 5, "Принципы автоматизации тестирования".

Последним типом шаблонов является антишаблон [AP]. Такие **запахи тестов** (test smells) соответствуют повторяющимся проблемам. Запахи описываются в виде наблюдаемых симптомов и реальных причин таких симптомов. **Запахи кода** (code smells)

впервые были описаны в книге Мартина Фаулера [Ref], а в отношении тестов на базе xUnit они были впервые описаны в докладе на конференции XP2001 [RTC]. Вместе с запахами тестов приводятся ссылки на шаблоны, которые позволяют от них избавиться, а также шаблоны (кто-то может назвать их антишаблонами), которые с большей вероятностью приведут к появлению такого запаха (некоторые шаблоны и запахи даже имеют одинаковые названия). Кроме того, запахи подробно рассматриваются в части II, "Запахи тестов".

Формат шаблона

Здесь приводится авторское описание шаблонов. Сами шаблоны существовали еще до написания книги, так как перед превращением в шаблон они были изобретены как минимум тремя независимыми разработчиками автоматизированных тестов. Книга написана с целью сделать знание более удобным для распространения, и для этого пришлось выбрать формат описания шаблонов.

Описания могут иметь несколько вариантов. Одни имеют жесткую структуру с большим количеством заголовков, помогающих читателю ориентироваться в разделах. Другие имеют более литературный формат, но более сложны в использовании в виде справочника.

Мой формат шаблонов

Мне действительно понравились работы Мартина Фаулера, в основном тем, что связано с форматом описания шаблонов. Как говорится, "имитация — искренняя форма лести", поэтому данный формат шаблонов был скопирован с минимальными модификациями.

Описание начинается с постановки задачи, общей информации и диаграммы. В выделенном курсивом абзаце описывается конкретная проблема, решаемая с помощью шаблона. Ее можно описать вопросом: "Как сделать...?" В абзаце, выделенном полужирным, описывается суть шаблона в одном-двух предложениях, а диаграмма обеспечивает визуальное представление. Текст после диаграммы содержит причины применения шаблона и разделы "Проблема" и "Контекст" из традиционного формата шаблона. Ознакомившись с этим разделом, читатель может принять решение о необходимости более детального изучения шаблона.

В следующих трех разделах формата приводится подробная информация о шаблоне. В разделе "Как это работает" описываются структура шаблона и принципы работы. Кроме того, приводится описание результирующего контекста, если существует несколько способов реализации важного аспекта шаблона. Этот раздел соответствует разделам "Решение" и "Следовательно" традиционного формата описания. В разделе "Когда это использовать" перечислены ситуации, когда имеет смысл применять шаблон. Этот раздел соответствует разделам "Проблема", "Вызывает", "Контекст" и "Похожие шаблоны" традиционного формата. Кроме того, раздел включает в себя описание результирующего контекста, если он может повлиять на решение о применении шаблона. Также перечислены запахи тестов, при которых рекомендуется применение шаблона. В разделе "Замечания по реализации" описываются особенности реализации конкретного шаблона. Подразделы в этом разделе соответствуют ключевым компонентам шаблона или вариантам реализации.

Большинство конкретных шаблонов имеют по три дополнительных раздела. Раздел "Мотивирующий пример" содержит пример тестового кода до применения шаблона. В разделе "Пример: (Имя шаблона)" показан код теста после применения шаблона. В разделе "Замечания по рефакторингу" приводится более подробная информация по переходу от кода из раздела "Мотивирующий пример" к коду из раздела "Пример: (Имя шаблона)".

Если существует дополнительная информация о данном шаблоне, описание может содержать раздел "Источники дополнительной информации". Раздел "Известные применения" содержит примечательные моменты конкретного теста. Конечно, большинство шаблонов использовались в множестве систем, поэтому указывать для каждого шаблона примеры использования будет лишней тратой времени.

Если существует несколько связанных одна с другой методик, обычно они описываются в виде шаблона с вариациями. Если вариации представляют собой различные способы реализации одного и того же фундаментального шаблона (т.е. решения одной проблемы одним общим способом), отличия в реализациях описываются в разделе "Замечания по реализации". Если отличие является основной причиной использования шаблона, оно рассматривается в разделе "Когда это использовать".

Исторические шаблоны и запахи

Значительных усилий стоило создать достаточно выразительный список пакетов и запахов, по возможности сохраняя исторические названия. Часто историческое название указывается в качестве псевдонима шаблона или запаха. В некоторых случаях историческое название имеет смысл рассматривать как вариант большего шаблона. В таком случае название указывается в разделе "Замечания по реализации".

Многие исторические запахи не прошли "проверку нюхом", т.е. они описывали главную причину, а не симптом. ("Проверка нюхом" основана на истории из книги *Рефакторинг*, в которой Кент Бек спрашивает бабушку: "Как узнать, что пришло время менять пеленку?" "Если воняет, меняй!" — ответила она. Названия запахов происходят от "вони", а не от ее причины.) Если историческое название описывает причину, а не симптом, оно перемещено в соответствующий симптоматический запах, как специальная вариация под названием "Причина". Хорошим примером кажется *таинственный гость* (Mystery Guest).

Ссылки на шаблоны и запахи

Определенных усилий потребовало создание ссылок на шаблоны и запахи, особенно создание исторические названия. Хотелось использовать как исторические, так и новые агрегированные названия, в зависимости от контекста. В электронной версии книги для этой цели применяются гиперссылки. Но в печатной версии в качестве ссылки используется номер страницы. Если ссылка указывает на вариант шаблона или причину запаха, в первый раз указывается имя агрегированного шаблона или запаха. Обратите внимание, что вторая ссылка на причину (*таинственный гость*, Mystery Guest) запаха *непонятный тест* (Obscure Test) указывается без имени запаха, а ссылка на другие причины *непонятного теста* (Obscure Test), например *неуместная информация* (Irrelevant Information,) включает в себя имя агрегированного запаха, но не включает номер страницы.

Рефакторинг

Рефакторинг является относительно новой концепцией в разработке программного обеспечения. Хотя модификация существующего кода требовалась всегда, рефакторинг является строго формализованным подходом к модификации дизайна без изменения поведения кода. Рефакторинг тесно связан с автоматизированным тестированием, так как очень сложно выполнять рефакторинг без страховочной сети тестов, которая позволяет доказать целостность кода.

Многие интегрированные среды разработки имеют встроенную поддержку рефакторинга. Большинство из них обеспечивают автоматизацию хотя бы нескольких операций, описанных в книге Мартина Фаулера [Ref]. К сожалению, эти инструменты не сообщают, когда и зачем выполнять рефакторинг. Для этого придется приобрести саму книгу. Также обязательно нужно прочитать книгу Джошуа Кериевски [RtP].

Рефакторинг тестов несколько отличается от рефакторинга работающего кода, так как для автоматизированных тестов не существует автоматизированных тестов! Если тест завершается неудачно после рефакторинга, может ли причиной быть ошибка во время рефакторинга? Если тест проходит успешно после рефакторинга, можно ли быть уверенным, что тест завершится неудачно в соответствующей ситуации? Для решения этой проблемы используются очень консервативные подходы к рефакторингу тестов. Кроме того, применяя подходящую стратегию тестирования (как показано в главе 6, “Стратегия автоматизации тестирования”), можно обойтись без внесения значительных модификаций в тесты.

Основное внимание здесь уделяется цели рефакторинга, а не механизмам. Краткое описание рефакторинга приводится в приложении А, но не это основная тема. Шаблоны сами по себе достаточно новы, чтобы не успело сформироваться общее мнение об их именовании, содержимом или применимости, тем более нет единого мнения о лучших способах рефакторинга. Наличие нескольких отправных точек для каждого объекта рефакторинга (шаблона) еще больше усложняет ситуацию. Попытка привести подробные инструкции по рефакторингу сделает эту, и так большую, книгу еще большей.

Предположения

При написании книги предполагалось, что читатель знаком с объектной технологией (так называемым объектно-ориентированным программированием). Объектная технология стала необходимым условием для роста популярности автоматизированного модульного тестирования. Это не значит, что тесты в процедурных или функциональных языках невозможны, но использование таких языков может усложнить тестирование (по крайней мере, все будет происходить немного по-другому).

Каждый человек имеет свой стиль обучения. Одним нужна общая картина с последующим спуском к достаточно подробным деталям. Другие могут понять детали и не нуждаются в общей картине. Третьим достаточно услышать или прочитать слова. Четвертым нужны визуальные представления концепций. Пятым изучение идей программирования лучше всего удается при чтении кода. Была сделана попытка приспособиться ко всем стилям обучения, поэтому везде, где это возможно, приводятся общие описания, подробные описания, примеры кода и иллюстрации. Эти элементы можно пропустить, если они не соответствуют выбранному вами стилю изучения.

Терминология

В этой книге собрана терминология из двух проблемных областей: разработки и тестирования программного обеспечения. В результате часть терминологии будет незнакома некоторым читателям. Встретив непонятный термин, желательно найти его в словаре терминов. Но один или два термина будут приведены здесь, так как их понимание является обязательным условием для понимания большей части материала данной книги.

Терминология тестирования

Разработчикам программного обеспечения термин *тестируемая система* (system under test — SUT) может показаться незнакомым. Означает он то, что в данный момент тестируется. При создании модульных тестов тестируемой системой является тестируемый класс или метод (методы). При создании клиентских тестов тестируемой системой, скорее всего, является целое приложение (или одна из его основных подсистем).

Любая часть приложения или создаваемой системы, не включенная в тестируемую систему, все равно может потребоваться для работы тестов, так как она вызывается тестируемой системой или предоставляет данные, необходимые тестируемой системе во время тестирования. Вызываемая часть называется *вызываемым компонентом* (depended-on component — DOC). И вызываемый компонент, и предоставляющая данные часть являются элементами тестовой конфигурации (рис. 1).

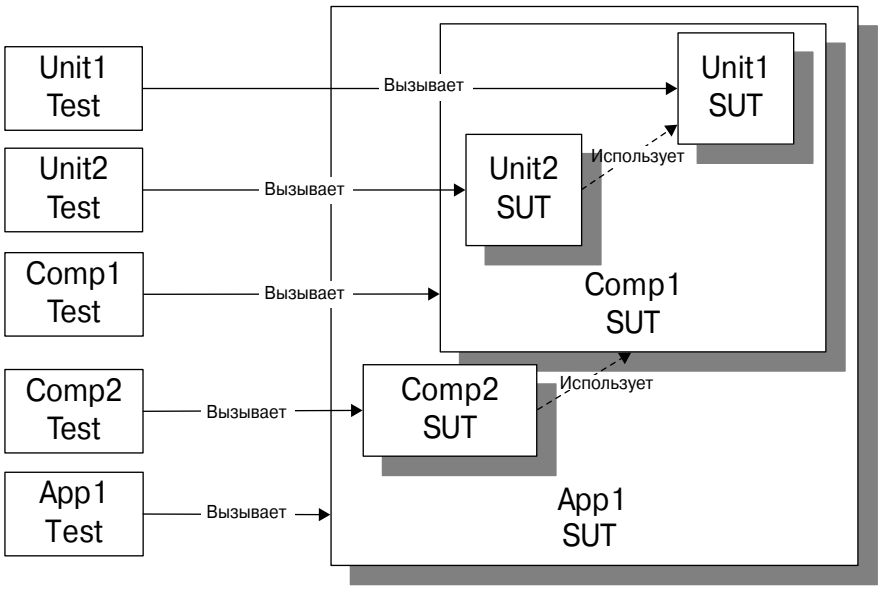

Рис. 1. Набор тестов. Каждому тесту соответствует своя тестируемая система. Приложение, компонент или модуль является тестируемой системой только относительно конкретного набора тестов. Некоторые тестируемые системы играют роль вызываемого компонента для других тестируемых систем

Зависящая от языка терминология xUnit

Хотя в этой книге содержатся примеры на основе различных языков и различных продуктов из семейства xUnit, очень часто используется продукт JUnit, так как многие знакомы с этой инфраструктурой. Существует несколько практически полных переводов JUnit на другие языки с минимальными модификациями имен классов и методов (в соответствии с особенностями используемого языка). Не всегда реализации оказываются полными. В таком случае обратитесь к приложению Б, "Терминология xUnit", в котором приводится соответствие между терминами в разных реализациях xUnit.

Использование Java как основного языка для примеров означает, что в некоторых случаях будут использоваться названия методов JUnit и не будут приводиться соответствующие названия из других инфраструктур xUnit. Например, в обсуждении может указываться имя метода `assertTrue` (JUnit) без указания, что эквивалентом в NUnit является `Assert.IsTrue`, в SUnit — `should:`, а в VbUnit — `verify`. Предполагается самостоятельное выполнение замены названий методов для SUnit, VbUnit, Test::Unit и других эквивалентов. Указывающих намерение названий методов JUnit должно быть достаточно для обсуждения интересующей тематики.

Примеры кода

Примеры кода всегда являются проблемой. Примеры из реальных проектов чаще всего слишком велики и связаны соглашением о неразглашении, запрещающим публикацию. "Игрушечные программы" не вызывают уважения, так как "они не настоящие". В данной книге практически нет выбора, кроме как использовать "игрушечные программы", но автором были приложены все усилия, чтобы сделать их максимально похожими на реальные проекты.

Практически все приведенные здесь примеры кода взяты из настоящего компилируемого и выполняемого кода, а значит, они не должны (стучим по дереву) содержать ошибок компиляции, если их не внесли в процессе редактирования. Большинство примеров на языке Ruby взято из системы публикации на основе XML, которая использовалась для подготовки данной книги. Примеры на языках Java и C# взяты из учебных материалов, которые компания ClearStream использует для обучения клиентов.

Несколько языков использовались для иллюстрации универсальной применимости инфраструктуры xUnit. В одних случаях конкретный шаблон продиктовал использование конкретного языка. Это может быть связано как с особенностями языка, так и с особенностями конкретной реализации инфраструктуры xUnit. В других случаях выбор языка продиктован доступностью сторонних расширений для конкретной реализации xUnit. Если особенностей нет, по умолчанию используется язык Java или C#, так как большинство разработчиков знакомы с ними как минимум на уровне чтения уже написанного кода.

Форматирование кода для книги оказалось отдельной сложной задачей, так как ширина строки составляет всего 65 символов. Некоторые имена переменных и классов сокращены для уменьшения количества переносов строк. Кроме того, были использованы определенные соглашения по поводу переноса строк для уменьшения вертикального размера примеров. Утешением должно служить то, что реальный тестовый код будет выглядеть намного более коротким, чем здесь, из-за меньшего количества переносов строк!

Описание с помощью диаграмм

"Лучше один раз увидеть, чем сто раз услышать". По возможности вместе с каждым тестом приводится диаграмма шаблона или запаха. В основном диаграммы базируются на языке UML (Unified Modeling Language), но несколько отступлений от стандарта позволили сделать их более выразительными. Например, используются символы агрегации (ромб) и наследования (треугольник) диаграммы классов UML, но на одной диаграмме одновременно используются классы и объекты, а также показаны ассоциации и взаимодействие объектов. Большая часть формата диаграмм описывается в главе 19, "Базовые шаблоны xUnit". Просмотрите эту главу хотя бы из-за рисунков.

Хотя предполагалось, что формат диаграмм "интуитивно понятен", стоит обратить внимание на несколько соглашений. Объекты имеют тени. Классы и методы их не имеют. Классы имеют прямые углы в соответствии с UML, методы имеют скругленные углы. Большие восклицательные знаки представляют *утверждение* (assertion) — потенциальное *неудачное завершение теста* (test failure). Символ звезды описывает ошибку или исключение. Облако обозначает тестовую конфигурацию. Центральный элемент диаграммы всегда выделен более жирными линиями и более темными тенями. В результате, сравнив две диаграммы, можно определить, что означает каждая из них.

Ограничения

При использовании шаблонов не забывайте, что автор не мог столкнуться со всеми проблемами автоматизации тестов и найти все решения этих проблем. Возможно, существуют лучшие способы их решения. Приведенные здесь решения просто работают для автора и его коллег. Любой совет принимайте с определенной долей сомнения!

Надеюсь, эти шаблоны станут отправной точкой для написания хороших, качественных автоматизированных тестов. При удачном стечении обстоятельств можно избежать многих ошибок, которые были допущены нами при первых попытках. Возможно, вы сумеете изобрести еще лучшие способы автоматизации тестов.