Введение

Название этой книги, *Адаптивный код*, является хорошим описанием итога применения принципов в книге: способность кода приспосабливаться к любому новому требованию или непредвиденному сценарию без необходимости в значительной переработке. Цель книги — собрать в одном месте множество текущих наилучших практик из мира программирования на C# с использованием Microsoft .NET Framework. Хотя некоторое содержимое раскрывается в других книгах, они либо сильно сконцентрированы на теории, либо не ориентированы на специфику разработки для .NET.

Программирование может быть медленным процессом. Если ваш код адаптивный, то вносить в него изменения будет быстрее, легче и с меньшим числом ошибок, чем в случае работы с кодовой базой, которая препятствует изменениям. Как известно каждому разработчику, требования подвержены изменениям. Управление изменениями — это основной фактор, отличающий успешные программные проекты от проектов, которые потерпели неудачу. Разработчики могут реагировать на изменения требований многими методами с двумя противостоящими точками зрения, который подчеркивают континуум, лежащий между ними.

Во-первых, разработчики могут придерживаться *жесткой* политики. При таком подходе от процесса разработки до проектирования классов проект оказывается негибким, как если бы он был реализован лет 50 тому назад с применением перфокарт. Заметными виновниками того, что ПО не может свободно изменяться, являются каскадные методологии. Их определение анализа, проектирования, реализации и тестирования как отдельных и однонаправленных стадий затрудняет — или, во всяком случае, делает дорогостоящим — изменение требований заказчиками после начала реализации. Следовательно, код совершенно не *нуждается* в построении с расчетом на изменение: процесс почти полностью запрещает внесение изменений.

Во-вторых, разработчики могут придерживаться *адаптивной* политики. Гибкие процессы — не просто альтернатива таким жестким методологиям, это реакция на них. Цель гибких процессов заключается в том, чтобы принять изменение как необходимую часть контракта между заказчиком и разработчиком. Если заказчики хотят изменить что-то в продукте, за который платят, то временны*е* и финансовые расходы должны соотноситься с размером изменения, а не со стадией процесса, которая выполняется в текущий момент.

В отличие от разработки физических объектов разработка ПО имеет дело с пластичным материалом: исходным кодом. В ходе строительства дома кирпичи и раствор буквально сливаются друг с другом. Расходы, которые влечет за собой изменение проекта дома, неизбежно связаны с завершением этапа строительства. Если проект пока не начался, т.е. он все еще в чертежах, тогда изменение будет относительно недорогим. Если окна вставлены, электропроводка подключена, а сантехника установлена, то перенос ванной комнаты с верхнего этажа вниз поближе к кухне может оказаться непомерно дорогостоящим. В случае кода перемещение функциональных средств и переделка навигации в пользовательском интерфейсе по идее не должны быть настолько затратными. К сожалению, это не всегда так. Часто такие изменения становятся невозможными только лишь по причине большого расхода времени. В значительной степени это результат неспособности кода к адаптации.

Кто должен читать эту книгу

Книга призвана ликвидировать разрыв между теорией и практикой. Во время ее написания я представлял себе читателя как программиста, ищущего более практичные примеры применения паттернов проектирования, принципов SOLID, модульного тестирования, рефакторинга и т.д.

Способные программисты средней квалификации, желающие восполнить пробелы в своих знаниях либо имеющие сомнения и вопросы о том, как наилучшие практики индустрии сочетаются друг с другом, извлекут наибольшую пользу от этой книги, главным образом потому, что повседневные реалии программирования редко совпадают с простыми примерами или теорией.

Большинство принципов SOLID теперь понимают, но запутанные аспекты принципа открытости/закрытости (глава 8) и принципа подстановки Лисков (глава 9) осмыслены не до конца. Даже опытные программисты временами не в полной мере осознают преимущества, обеспечиваемые внедрением зависимостей (глава 12). Аналогично часто игнорируется гибкость — способность к адаптации — кода, реализованного с использованием интерфейсов (глава 4).

Множество материалов позволит начинающим разработчикам узнать, какие аспекты распространенных паттернов и практик благоприятны, а какие проблематичны в долгосрочном плане. Разные кодовые базы, которые я встречал во время проведения консультаций, имели много общих проблем. Основной вопрос в том, что код *почти* достиг цели в плане способности к адаптации, но ему нужен просто легкий толчок в правильном направлении, чтобы стать значительно более адаптивным к изменениям. В частности, в коде застопорившихся проектов преобладают антипаттерны “Антураж” (Entourage), рассмотренный в главе 11, и “Локатор служб” (Service Locator), описанный в главе 12. В этой книге приводятся практические альтернативы вместе с их обоснованиями.

Допущения

В идеале вы должны иметь некоторый практический опыт программирования на языке, синтаксически похожем на C#, таком как Java или C++. Вы также должны хорошо понимать основные концепции процедурного программирования вроде условного ветвления, циклов и выражений. Кроме того, вы должны знать концепции объектно-ориентированного программирования с применением классов и хотя бы немного понимать интерфейсы.

Эта книга может быть не для вас, если...

Эта книга может быть не для вас, если вы только начали изучать программирование. В книге раскрываются сложные темы, освоение которых требует четкого понимания фундаментальных концепций объектно-ориентированного программирования, подобных классам и интерфейсам, в дополнение к управляющим структурам, таким как операторы `if` и циклы.

Организация этой книги

Книга состоит из четырех частей, каждая из которых основана на предыдущей. Тем не менее, книгу можно также читать не по порядку. В каждой главе подробно рассматривается независимая тема и при необходимости приводятся перекрестные ссылки.

Изменения во втором издании

Для читателей первого издания есть веские причины приобрести это обновленное издание:

- добавлена новая глава по Kanban, дополняющая главу по Scrum;
- тестирование и рефакторинг теперь занимают отдельные главы, так что раскрывается больший набор вопросов;
- глава, посвященная инверсии зависимостей, переписана для включения раздела по проектированию абстракций;
- добавлена новая глава, посвященная связанности и сцеплению;
- примеры кода обновлены до версий C# 7.0 и .NET 4.6.2 в Microsoft Visual Studio 2017;
- все остальные главы отредактированы и уточнены.

Часть I. Инфраструктуры гибкой разработки

В этой части предлагается введение в контекст, который делает настолько важным написание адаптивного кода. В ней охвачены две гибкие инфраструктуры, Scrum и Kanban, чтобы показать, что изменения неизбежны и к ним должны приспосабливаться процесс поставки и код.

Глава 1. Введение в Scrum

В этой главе представлена гибкая инфраструктура Scrum, которая предназначена для поставки ПО. Она включает исчерпывающий обзор артефактов, ролей, метрик и стадий проекта Scrum. Кроме того, в главе раскрываются проблемы, с которыми часто сталкиваются команды, использующие Scrum, а также способы их распознавания и исправления.

Глава 2. Введение в канбан [новая]

Чтобы дополнить главу по Scrum, второе издание книги знакомит с канбан как альтернативной гибкой инфраструктурой, которая больше сосредоточена на непрерывном потоке поставляемых функциональных средств. В этой главе рассматриваются артефакты Kanban и предлагаются примеры для выявления проблем в потоке поставки.

Часть II. Основы адаптивного кода

Эта часть закладывает основы для построения ПО адаптивным методом. В ней раскрывается способность к адаптации на всех уровнях кода и акцентируется внимание на деталях интерфейсов, паттернов проектирования, рефакторинга и модульного тестирования.

Глава 3. Зависимости и разделение на уровни

В данной главе исследуются зависимости и разделение на архитектурные уровни. Код может быть адаптивным, только если это позволяет структура решения. Описаны различные типы зависимостей: основные, сторонние и инфраструктуры. Объясняется, как управлять и организовывать зависимости, от антипаттернов (которых следует избегать) до паттернов (которые должны приниматься). Также рассматриваются такие более сложные темы, как аспектно-ориентированное программирование и асимметричное разделение на уровни, обеспечивающее дополнительную глубину.

Глава 4. Интерфейсы и паттерны проектирования

В современной разработке с применением .NET интерфейсы вездесущи. Однако их часто некорректно используют, неправильно понимают и применяют в неподходящих местах. В этой главе демонстрируются некоторые более распространенные и практически полезные паттерны проектирования, а также исследуется разносторонний характер интерфейсов. Выводя читателя за рамки простого извлечения интерфейса, в главе показано, как интерфейсы могут совершенствоваться разнообразными путями для решения задачи. Примеси, утиная типизация и текучесть интерфейсов дополнительно подчеркивают разносторонность этого ключевого инструмента в арсенале программиста.

Глава 5. Тестирование [новая]

Модульное тестирование представляет собой не допускающее обсуждений предварительное условие в любом программном проекте. Модульное тестирование тесно связано с рефакторингом. Работая в унисон, эти две практики позволяют выпускать адаптивный код. Без сетки безопасности в виде модульных тестов рефакторинг предрасположен к ошибкам; без рефакторинга код становится громоздким, жестким и трудным для понимания. В этой главе приводится первоначально непритязательный пример модульного тестирования, который расширяется с целью использования более сложных, но практичных средств вроде текучих утверждений, разработки через тестирование и имитации.

Глава 6. Рефакторинг [новая]

Рефакторинг означает улучшение структуры кода после того, как он был написан. В этой главе рассмотрен пример реалистичного рефакторинга, который повышает читабельность и удобство сопровождения исходного кода. Код обязан иметь тесты, чтобы его можно было безопасно подвергать рефакторингу, но в главе также показано, каким образом добавлять тесты к существующему коду для обеспечения безопасного рефакторинга.

Часть III. Код SOLID

Эта часть построена на основе части II. Каждая глава посвящена исследованию одного из принципов SOLID. Акцент в главах делается на практических примерах для внедрения принципов, а не только на теоретических причинах их важности. За счет помещения каждого примера в реалистичный контекст главы в данной части книги ясно демонстрируют полезность принципов SOLID.

Глава 7. Принцип единственной обязанности

В этой главе показано, как обеспечить выполнение принципа единственной обязанности на практике, используя паттерны “Декоратор” (Decorator) и “Адаптер” (Adapter). Последствием применения данного принципа является увеличение количества классов и сокращение их размеров. В главе демонстрируется, что по контрасту с монолитными классами, которые предоставляют обширные функциональные возможности, такие мелкие классы больше сосредоточены на решении лишь небольших частей крупной задачи. Именно в совокупности эти классы становятся чем-то большим, нежели сумма их частей.

Глава 8. Принцип открытости/закрытости

Принцип открытости/закрытости (OCP) формулируется просто, но может оказывать значительное влияние на код. Он отвечает за гарантирование того, что код, следующий принципам SOLID, допускает только дополнение, но не редактирование. В главе также обсуждается концепция предсказуемых изменений в связи с принципом OCP и выясняется, как она может помочь разработчикам идентифицировать точки расширения для повышения степени адаптации.

Глава 9. Принцип подстановки Лисков

В этой главе показаны положительные эффекты, которые проявляются в коде из-за применения принципа подстановки Лисков, в частности, тот факт, что рекомендации помогают соблюсти принцип открытости/закрытости и предотвратить непредусмотренные последствия от изменения. Контракты — через предусловия, постусловия и инварианты данных — рассматриваются с использованием инструментов Microsoft Code Contracts. В главе также описаны руководящие указания относительно создания подтипов, такие как ковариантность, контравариантность и инвариантность, наряду с негативными последствиями нарушения этих указаний.

Глава 10. Разделение интерфейса

В этой главе показано, что уменьшаться должны не только классы, но и интерфейсы, которые часто оказываются слишком большими. Разделение интерфейса — простая практика, которой нередко пренебрегают; в главе демонстрируются преимущества ограничения интерфейсов до наименьших возможных размеров вместе с выгодами работы с небольшими интерфейсами. В ней также исследуются причины, которые могут подтолкнуть к разделению интерфейсов, такие как потребность клиента и архитектурная потребность.

Глава 11. Инверсия зависимостей [новая]

В этой главе объясняется важность управления зависимостями и показано, как добиться развязанных модулей. Обсуждается распространенный антипаттерн и рассматривается предпочтительная альтернатива. Кроме того, в главе прорабатывается пример абстракции, которая доказывает, что интерфейсы — только часть решения.

Часть IV. Применение адаптивного кода

Эта часть является логическим продолжением глав о принципах SOLID, представляя теорию и практики, которые объединяют все вместе.

Глава 12. Внедрение зависимостей

Эта глава посвящена связующему механизму, который содействует работе остальных средств, описанных в книге. Без внедрения зависимостей многое было бы невозможным — оно действительно настолько важно. В главе содержится введение во внедрение зависимостей и производится сравнение разных методов его реализации. Кроме того, обсуждаются такие темы, как управление временем жизни объектов, контейнеры инверсии управления, распространенные антипаттерны, связанные с местоположением служб, и идентификация корней композиции и распознавания.

Глава 13. Связанность, сцепление и соразвитие [новая]

Все паттерны проектирования помогают удерживать связанность низкой, а сцепление высоким. В этой главе определяются связанность и сцепление, а также вводится измерение силы связывания через соразвитие.

Приложение

В приложении приведено очень краткое введение в систему управления исходным кодом Git, которая должна, по меньшей мере, позволить вам загрузить код из GitHub и скомпилировать его в Visual Studio 2017. Приложение не задумывалось как исчерпывающее руководство по Git — для этого доступно несколько великолепных источников, включая официальный учебник по Git:

`http://git-scm.com/docs/gittutorial`

Другие источники ищите в Интернете.

Соглашения, принятые в этой книге

При оформлении книги использовалось несколько соглашений. В основном они являются стандартными для изданий Microsoft Press, но заранее объяснить их не помешает.

Листинги кода

Листинги кода включаются там, где они уместны, и при необходимости снабжаются заголовками, как показано в листинге 0.1.

Листинг 0.1. Это листинг кода; они часто встречаются в книге

```
public void MyService : IService
{
}
```

Всякий раз, когда нужно привлечь ваше внимание к определенной части кода, например, к изменениям, внесенным в предыдущий пример, применяется выделение полужирным.

Замечания и врезки

Замечания используются для небольших отклонений от темы, таких как заметки или предупреждения, тогда как врезки содержат более крупные отступления. Ниже приведены примеры.

На заметку! Это замечание для читателя. Элементы такого рода содержат небольшие фрагменты информации, которые относятся к основному содержанию, но имеют какое-то дополнительное значение.

Врезка

Хотя данная врезка оказалась неизбежно короткой, обычно врезки приберегаются для более длинных обсуждений на темы, которые не имеют прямого отношения к основной теме.

Рисунки

Иногда одного объяснения, каким бы оно ни было цветистым, недостаточно. В таких случаях предоставляется рисунок. Все диаграммы были созданы в Microsoft Visio 2013 без применения тем, чтобы добиться высокой контрастности и сфокусироваться исключительно на изображении. Снимки экрана были получены с использованием темы с высокой контрастностью.

Системные требования

Для проработки примеров, рассмотренных в этой книге, вам понадобятся следующие компоненты:

- Visual Studio 2015 или последующей версии, любая редакция;
- Microsoft SQL Server 2008 Express Edition или последующей версии (выпуск 2008 либо R2) с SQL Server Management Studio 2008 Express или последующей версии;
- подключение к Интернету для загрузки кода примеров.

В зависимости от имеющейся конфигурации Windows для установки или конфигурирования продуктов Visual Studio и SQL Server могут потребоваться права локального администратора.

Загрузка кода примеров

По мере возможности я стремился к тому, чтобы листинги кода были частью более крупного примера, который можно было бы запускать либо как автономное приложение, либо как модульный тест. Я написал много простых модульных тестов с применением MSTest, так что какое-то внешнее средство прогона тестов для них не нужно, но более сложные тесты создавались с использованием NUnit. Для написания всего кода я применял Visual Studio 2017 Professional. Несмотря на то что некоторый код был написан с использованием версии Community, весь код компилировался и тестировался в полной версии Visual Studio. Там, где удавалось, я не применял средства, которые не доступны в версиях Community продукта Visual Studio 2017, но для ряда тем это оказывалось попросту невозможным. Читателям, желающим работать с таким кодом, придется установить платную версию.

Сам код примеров доступен в GitHub по следующему адресу:

```
https://github.com/AdaptiveCode/AdaptiveCode
```

В приложении вы найдете краткие объяснения по использованию Git и по организации хранилища `Adaptive Code`.

Чтобы сделать комментарий, где я его, возможно, увижу, обратитесь в хранилище GitHub. Вы также можете читать меня в Твиттере:

```
@garymcleanhall
```

Ждем ваших отзывов!

Вы, читатель этой книги, и есть главный ее критик. Мы ценим ваше мнение и хотим знать, что было сделано нами правильно, что можно было сделать лучше и что еще вы хотели бы увидеть изданным нами. Нам интересны любые ваши замечания в наш адрес.

Мы ждем ваших комментариев и надеемся на них. Вы можете прислать нам бумажное или электронное письмо либо просто посетить наш веб-сайт и оставить свои замечания там. Одним словом, любым удобным для вас способом дайте нам знать, нравится ли вам эта книга, а также выскажите свое мнение о том, как сделать наши книги более интересными для вас.

Отправляя письмо или сообщение, не забудьте указать название книги и ее авторов, а также свой обратный адрес. Мы внимательно ознакомимся с вашим мнением и обязательно учтем его при отборе и подготовке к изданию новых книг.

Наши электронные адреса:

E-mail: `info@dialektika.com`
WWW: `http://www.dialektika.com`

Наши почтовые адреса:

в России: 195027, Санкт-Петербург, Магнитогорская ул., д. 30, ящик 116
в Украине: 03150, Киев, а/я 152