

Содержание

Об авторах	11
Предисловие	13
Глава 1. Введение	26
1.1. Что такое обслуживаемость?.....	27
Четыре вида обслуживаемости программного обеспечения.....	27
1.2. Почему так важна обслуживаемость?.....	28
Обслуживаемость значительно влияет на деловую сторону вопроса.....	28
Обслуживаемость обеспечивает улучшение других качественных характеристик.....	29
1.3. Три принципа, на которых основаны рекомендации.....	30
Принцип 1: рекомендации должны быть простыми.....	30
Принцип 2: применение рекомендаций с самого начала и значимость вклада каждого разработчика.....	31
Принцип 3: не все отступления от рекомендаций дают одинаковый отрицательный эффект.....	31
1.4. Заблуждения относительно обслуживаемости.....	32
Заблуждение: обслуживаемость зависит от языка программирования.....	32
Заблуждение: обслуживаемость зависит от прикладной области.....	33
Заблуждение: обслуживаемость гарантирует отсутствие ошибок.....	33
Заблуждение: обслуживаемость оценивается одной из двух альтернатив.....	34
1.5. Рейтинг обслуживаемости.....	34
1.6. Обзор рекомендаций по улучшению обслуживаемости.....	36
Глава 2. Пишите короткие блоки кода	38
2.1. Мотивация.....	41
Короткие блоки кода проще тестировать.....	41
Короткие блоки кода проще анализировать.....	42
Короткие блоки кода проще повторно использовать.....	42
2.2. Как применять рекомендацию.....	42
При написании нового блока кода.....	43
При добавлении в блок новых функциональных возможностей.....	44

	Два метода рефакторинга для приведения кода в соответствие с рекомендацией	45
2.3.	Типичные возражения против коротких блоков кода	51
	Возражение: увеличение количества блоков кода плохо сказывается на производительности	51
	Возражение: разделение кода ухудшает читаемость	51
	Рекомендация препятствует надлежащему форматированию кода	52
	Этот блок кода невозможно разделить	53
	Разделение блоков кода не дает заметных преимуществ	54
2.4.	Дополнительные сведения	55
Глава 3. Пишите простые блоки кода		57
3.1.	Мотивация	64
	Простые блоки проще изменять	64
	Простые блоки проще тестировать	64
3.2.	Как применять рекомендацию	64
	Цепочки условий	65
	Вложенность	67
3.3.	Типичные возражения против создания простых блоков кода	70
	Возражение: высокая сложность неизбежна	70
	Возражение: разделение методов не уменьшает сложности	70
3.4.	Дополнительные сведения	71
Глава 4. Не повторяйте один и тот же код		73
	Виды дублирования	76
4.1.	Мотивация	78
	Код с дубликатами сложнее анализировать	78
	В дублированный код сложно вносить изменения	78
4.2.	Как применять рекомендацию	78
	Извлечение суперкласса	81
4.3.	Типичные возражения против исключения дублирования	84
	Копирование фрагментов из другой базы кода допустимо	84
	При незначительных изменениях дублирование неизбежно	85
	Этот код никогда не изменится	85
	Дублирование всех файлов допустимо в целях создания их резервных копий	86
	Модульные тесты защитят меня	86
	Дублирование строковых литералов неизбежно и совершенно безвредно	87
4.4.	Дополнительные сведения	87

Глава 5. Стремитесь к уменьшению размеров интерфейсов	90
5.1. Мотивация.....	92
Интерфейсы небольшого размера упрощают понимание и повторное использование кода.....	93
В методы с компактным интерфейсом проще вносить изменения.....	93
5.2. Как применять рекомендацию.....	93
5.3. Типичные возражения против сокращения размеров интерфейсов.....	98
Возражение: объекты параметров требуют определения конструкторов с большим количеством параметров.....	99
Преобразование интерфейсов большого размера не улучшает ситуацию.....	99
Фреймворки или библиотеки предоставляют интерфейсы с длинными списками параметров.....	99
5.4. Дополнительные сведения.....	100
 Глава 6. Разделяйте задачи на модули	 102
6.1. Мотивация.....	107
Небольшие слабо связанные модули позволяют разработчикам иметь дело с надежно изолированными частями системы.....	108
Небольшие слабо связанные модули упрощают навигацию по коду.....	108
Небольшие слабо связанные модули делают все области кода более понятными новым разработчикам.....	108
6.2. Как применять рекомендацию.....	109
Разделение классов по решаемым задачам.....	109
Соккрытие подробностей реализации за интерфейсами.....	110
Замена пользовательского кода библиотеками или фреймворками от сторонних производителей.....	114
6.3. Типичные возражения против разделения задач.....	114
Возражение: слабые связи вступают в конфликт с возможностью повторного использования.....	114
Возражение: интерфейсы языка C# не предназначены для ослабления связей.....	115
Возражение: высокая нагрузка на служебные классы неизбежна.....	115
Возражение: не все слабо связанные решения улучшают обслуживаемость.....	115

Глава 7. Избегайте тесных связей между элементами архитектуры.....	118
7.1. Мотивация.....	119
Слабая зависимость между компонентами обеспечивает изолированность его обслуживания	122
Слабая зависимость компонентов способствует разделению ответственности за обслуживание	122
Слабая зависимость компонентов упрощает тестирование.....	123
7.2. Как применять рекомендацию	123
Абстрактная фабрика как шаблон проектирования	124
7.3. Типичные возражения против устранения тесных связей компонентов.....	126
Возражение: зависимости между компонентами невозможно смягчить из-за тесного переплетения компонентов	126
Возражение: нет времени на исправление	127
Возражение: транзитный код просто необходим.....	127
7.4. Дополнительные сведения	128

Глава 8. Стремитесь к сбалансированности архитектуры компонентов.....	130
8.1. Мотивация.....	132
Хороший баланс компонентов упрощает поиск и анализ кода	132
Хорошо сбалансированные компоненты улучшают изолированность обслуживания	132
Хорошо сбалансированные компоненты позволяют распределять ответственность при обслуживании	133
8.2. Как применять рекомендацию	133
Выбор правильного концептуального уровня при распределении функциональных возможностей по компонентам	133
Последовательное применение разделения системы на основе анализа предметной области.....	134
8.3. Типичные возражения против стремления к сбалансированности компонентов	135
Возражение: системы с дисбалансом компонентов отлично работают	135
Возражение: запутанность связей между компонентами не позволяет их сбалансировать.....	135
8.4. Дополнительные сведения	136

Глава 9. Следите за размером базы кода	138
9.1. Мотивация.....	139
Проект с большой базой кода, скорее всего, обречен на неудачу.....	139

Большие базы кода труднее обслуживать.....	139
Большие системы отличаются высокой плотностью дефектов.....	141
9.2. Как применять рекомендацию	142
Функциональные меры.....	142
Технические меры.....	142
9.3. Типичные возражения против уменьшения размеров базы кода	144
Возражение: сокращение размера базы кода снижает продуктивность разработки	145
Возражение: выбранный язык программирования препятствует уменьшению объема кода	145
Возражение: сложность системы заставляет дублировать код.....	146
Возражение: разделение базы кода невозможно из-за архитектуры платформы.....	146
Возражение: разделение кода приводит к дублированию.....	147
Возражение: разделение базы кода невозможно из-за тесной связанности.....	147
Глава 10. Автоматизируйте тестирование	149
10.1. Мотивация.....	151
Автоматизация делает тестирование повторяемым	151
Автоматизированное тестирование увеличивает эффективность разработки.....	151
Автоматизированное тестирование делает код предсказуемым	151
Тесты документируют тестируемый код	152
Разработка тестов улучшает качество кода	152
10.2. Как применять рекомендацию	153
Начало работы с NUnit	154
Общие принципы разработки хороших модульных тестов.....	157
Оценка охвата для определения достаточности количества тестов.....	162
10.3. Типичные возражения против автоматизации тестов	164
Возражение: нам нужно и ручное тестирование.....	164
Возражение: мне не разрешается писать модульные тесты	164
Возражение: зачем тратить время на модульные тесты при низком текущем охвате ими?	165
10.4. Дополнительные сведения	165
Глава 11. Пишите чистый код	167
11.1. Не оставляйте следов	167
11.2. Как применять рекомендацию	168
Правило 1: не оставляйте после себя грязи на уровне блоков кода.....	168

Правило 2: не оставляйте после себя неудачных комментариев.....	169
Правило 3: не оставляйте после себя закомментированного кода.....	171
Правило 4: не оставляйте после себя неиспользуемого кода	172
Правило 5: не оставляйте после себя длинных идентификаторов	173
Правило 6: не оставляйте после себя таинственных констант	173
Правило 7: не оставляйте после себя плохую обработку исключений	175
11.3. Типичные возражения против написания чистого кода	175
Возражение: комментарии являются документацией	175
Возражение: обработка исключений увеличивает объем кода	176
Возражение: почему выбраны именно эти правила?	176
Глава 12. Дальнейшие действия	178
12.1. Применение рекомендаций на практике	178
12.2. Низкоуровневые (блоки кода) рекомендации имеют более высокий приоритет, если они противоречат высокоуровневым (компоненты) рекомендациям	179
12.3. Помните, что учитывается каждое действие	179
12.4. Передовой опыт разработки будет рассмотрен в следующей книге	180
Приложение А. Как в компании SIG оценивается обслуживаемость	181
Предметный указатель	184

Об авторах

Джуст Виссер (Joost Visser) – научный руководитель Software Improvement Group (SIG). В этой роли он отвечает за научное обоснование методов и инструментов, предлагаемых компанией SIG для количественной оценки и улучшения программного обеспечения. Кроме того, Джуст занимает должность профессора кафедры крупномасштабных программных систем в университете Неймегена (Нидерланды). Получил докторскую степень в области компьютерных наук в университете Амстердама и опубликовал более 100 статей по таким темам, как обобщенное программирование, трансформация программ, эффективные вычисления, качество программного обеспечения, а также эволюция программного обеспечения. Джуст считает разработку программного обеспечения социально-технической дисциплиной и убежден, что количественная оценка программного обеспечения имеет важное значение для успешности команды разработчиков и владельцев программной продукции.

Паскаль ван Экк (Pascal van Eck) присоединился к Software Improvement Group в 2013 году в роли консультанта по общим вопросам качества программного обеспечения. До прихода в SIG в течение 13 лет был доцентом кафедры информационных систем в университете Твенте (Нидерланды). Имеет докторскую степень в области компьютерных наук, полученную в университете Врийе (Амстердам), и опубликовал более 80 статей в таких областях, как архитектура корпоративных приложений, компьютерная безопасность и количественная оценка программного обеспечения. Паскаль является председателем программного комитета голландской национальной конференции по архитектуре цифрового мира.

После получения степени магистра в области программной инженерии в техническом университете Делфта в 2005 году **Роб ван дер Лик** (Rob van der Leek) присоединился к SIG в роли консультанта по качеству программного обеспечения. Работая в SIG, Роб ощущает себя врачом программного обеспечения. Как консультант он, опираясь на свои технические знания в области разработки программного обеспечения и программных технологий, дает клиентам советы по поддержанию их программных систем в хорошей форме. Кроме работы консультантом, Роб занимает ведущее положение

в команде внутренних разработок SIG. Эта команда разрабатывает и обслуживает аналитическое программное обеспечение компании. Амбициозной мечтой Робу служит желание оставить IT-индустрию в состоянии, хоть немногим лучшим, чем то, в котором она была, когда он в нее вошел.

Сильван Ригаль (Sylvan Rigal) работает в SIG консультантом по вопросам качества программного обеспечения с 2011 года и дает клиентам советы по вопросам управления программным обеспечением с 2008 года. Он помогает клиентам добиться снижения затрат на обслуживание программного обеспечения и повышения его безопасности за счет совершенствования процессов проектирования и разработки программ. Имеет степень магистра в области международного бизнеса Маастрихтского университета (Нидерланды). Как активный член команды безопасности программного обеспечения SIG обучает консультантов анализу рисков безопасности программного обеспечения. Когда Сильван не занят оценкой технического здоровья программ, он совершенствуется в бразильском джиу-джитсу, с удовольствием посещает рестораны Амстердама или путешествует по Азии. Вот примерно в таком порядке.

Гис Винхолдс (Gijs Wijnholds) присоединился к Software Improvement Group в 2015 году в качестве консультанта по программному обеспечению в области государственного управления. Помогает клиентам управлять их программными проектами, консультируя по вопросам развития и перевода технических рисков в стратегические решения. Гис получил степень бакалавра в области искусственного интеллекта в Утрехтском университете (Нидерланды) и степень магистра логики в университете Амстердама. Является экспертом по языку программирования Haskell и математической лингвистике.

Предисловие

В умении обходиться малым виден мастер.

Йоганн Вольфганг фон Гёте

После 15 лет консультирования по вопросам качества программного обеспечения мы в Software Improvement Group (SIG) усвоили два аспекта, касающихся обслуживаемости.

Во-первых, неудовлетворительная обслуживаемость является реальной проблемой в практике разработки программного обеспечения – разработчики тратят слишком много времени на обслуживание и исправление старого кода. Это оставляет меньше времени на наиболее плодотворную часть разработки программного обеспечения – написание нового кода. Наш собственный опыт, а также собранные нами данные свидетельствуют от том, что обслуживание исходного кода занимает, по крайней мере, в два раза больше времени, когда обслуживаемость находится на уровне ниже среднего, чем при уровне обслуживаемости выше среднего. С нашей методикой оценки обслуживаемости можно ознакомиться в приложении А.

Во-вторых, недостатки в обслуживаемости в значительной степени вызваны тривиальными просчетами в разработке, повторяемыми снова и снова. Следовательно, наиболее эффективный и действенный способ улучшить обслуживаемость заключается в исключении этих просчетов. Для этого не требуется никакого волшебства или чего-то из ряда вон выходящего. Сочетание относительно простых навыков, знаний плюс дисциплина и соответствующее окружение обеспечивают заметное повышение уровня обслуживаемости.

В компании SIG мы не раз сталкивались с системами, которые по существу являются непригодными для эксплуатации. В этих системах нет возможности исправить ошибки, а их функциональность не модифицируется или не расширяется, поскольку это требует слишком больших временных затрат и связано с огромными рисками. К сожалению, такое явление не редкость в современной IT-индустрии, но ведь это недопустимо.

Именно поэтому мы сформулировали 10 рекомендаций. Мы хотим поделиться знаниями и навыками со всеми практикующими разработчиками программного обеспечения, чтобы помочь им писать

обслуживаемый исходный код. Мы уверены, что после знакомства с этими 10 рекомендациями вы как разработчик программного обеспечения сможете писать обслуживаемый исходный код. После этого вам останется только сформировать окружение, в котором вы сможете применить эти навыки с максимальным эффектом, включая общие методики разработки, соответствующий инструментарий и многое другое. Основам среды разработки будет посвящена наша вторая книга «*Building Software Teams*».

Тема этой книги: Десять правил разработки обслуживаемого программного обеспечения

Изложенное в следующих главах применимо к любой системе разработки. Рекомендации касаются размеров блоков кода и количества параметров (методов в языке C#), числа точек ветвления, а также других свойств исходного кода. Это давно известные правила, о которых наверняка слышали многие программисты. В главах также приводятся примеры, как правило, в форме шаблонов рефакторинга кода, демонстрирующие применение этих рекомендаций на практике. Все примеры написаны на языке C#, но сами советы не зависят от используемого языка программирования. Восемь советов из 10 заимствованы из выработанных в SIG/TÜViT¹ критериев разработки для достижения уверенной обслуживаемости продукта² – комплекса характеристик для системной оценки обслуживаемости исходного кода.

Почему следует прочесть эту книгу

Каждая из взятых по отдельности рекомендаций, представленных в этой книге, хорошо известна. Действительно, многие популярные инструменты анализа кода выполняют проверку представлен-

¹ TÜViT является частью TÜV, всемирной организации контроля качества техники немецкого происхождения. Она специализируется на сертификации и консалтинге в области программного обеспечения, в частности в вопросах его безопасности.

² Более подробную информацию о критериях обслуживаемости можно найти на странице http://bit.ly/eval_criteria.

ных здесь рекомендаций. Например, инструменты Checkstyle (для Java, <http://checkstyle.sourceforge.net/>), StyleCop+ (для C#, <http://stylecopplus.codeplex.com/>), Pylint (для Python, <http://www.pylint.org/>), JSHint (для JavaScript, <http://jshint.com/>), RuboCop (для Ruby, <https://github.com/bbatsov/rubocop>) и PMD (охватывает несколько языков, включая C# и Java, <https://pmd.github.io/>) проверяют код на соответствие рекомендации, представленной в главе 2. Однако три особенности, перечисленные ниже, выделяют эту книгу из числа других, посвященных разработке программного обеспечения:

Мы отобрали 10 самых важных рекомендаций на основании собственного опыта

Инструменты контроля стиля и статического анализа кода обычно слишком сложны. Версия Checkstyle 6.9 выполняет проверку с использованием порядка 150 правил. Все они полезны, но по-разному влияют на обслуживаемость. Мы отобрали 10, имеющих наибольшее влияние на обслуживаемость. Наш выбор обосновывается ниже, во врезке «Почему именно эти десять конкретных рекомендаций?».

Мы научим применению этих 10 рекомендаций

Простое перечисление, *что* следует делать программисту и чего не следует, – это одно (а многие руководства ограничиваются только этим). А описание, *как* следовать рекомендациям, – это совсем другое. В этой книге каждая рекомендация сопровождается конкретными примерами создания соответствующего ей кода.

Мы приведем статистические данные и примеры, взятые из реальных систем

В компании SIG мы просмотрели *массу* исходного кода, созданного реальными программистами в реальных условиях. Для этого исходного кода характерен компромиссный подход. Мы поделимся результатами своих исследований и покажем, насколько реальный исходный код соответствует этим рекомендациям.

Кому адресована эта книга

Эта книга адресована разработчикам программного обеспечения, которые уже умеют программировать на C#. Таких разработчиков можно разбить на две группы. Первая включает разработчиков, которые получили всестороннюю подготовку в области информатики или программной инженерии (например, обучаясь по соответствующей специальности в колледже или университете). Этим разработчикам

наша книга поможет закрепить принципы, которым их обучали на вводных курсах программирования.

Вторая группа включает разработчиков, которые приступили к работе, не получив полноценного образования в области информатики или программной инженерии. Здесь имеются в виду разработчики, которые обучались самостоятельно или специализировались, учась в колледже или университете, в совершенно иной области, а затем сменили направление деятельности. Наш опыт показывает, что специалисты из этой группы, как правило, весьма слабо осведомлены в вопросах, выходящих за рамки синтаксиса и семантики используемого ими языка программирования. Именно на эту группу мы ориентировались при написании книги.

Почему именно эти десять конкретных рекомендаций?

В книге представлено 10 рекомендаций. Первые восемь напрямую связаны с так называемыми *системными свойствами* критериев оценки уверенной обслуживаемости готовых продуктов SIG/TŪViT, на которых основывается показатель обслуживаемости SIG. Из критериев оценки SIG/TŪViT мы выбрали показатели, которые:

- наиболее редко упоминаются;
- не зависят от технологии;
- легко измеряются;
- поддаются оценке в реальных системах корпоративного программного обеспечения.

В результате были отобраны восемь системных свойств, включенных в критерии оценки SIG/TŪViT. К ним мы добавили две процессуальные рекомендации (касающиеся чистоты кода и автоматизации), которые посчитали самыми важными и поддающимися непосредственному контролю.

Специалисты в области информатики и программной инженерии весьма преуспели в определении характеристик исходного кода. В зависимости от порядка подсчета были зарегистрированы десятки, если не сотни различных характеристик. Поэтому восемь используемых нами системных свойств – далеко не полный перечень характеристик обслуживаемости.

Но мы полагаем, что эти восемь характеристик SIG/TŪViT являются вполне адекватным и достаточным набором для оценки обслуживаемости, поскольку позволяют решить следующие задачи:

Характеристики, зависящие от технологии

Некоторые характеристики (например, глубина наследования) применимы только к исходному коду, привязанному к конкретным технологиям (например, только в объектно-ориентированных язы-

ках). В то время как доминирующий на практике объектно-ориентированный подход является далеко не единственным. Существует достаточно много исходного кода, не являющегося объектно-ориентированным (вспомните языки Cobol, RPG, С и Pascal), обслуживание которого также нужно оценить.

Характеристики, сильно коррелирующие с другими

Некоторые характеристики отличаются заметной корреляцией друг с другом. Примером может служить общее число точек ветвления в коде. Эмпирически доказано, что эта характеристика в значительной степени коррелирует с характеристикой, определяющей объем кода. Это значит, что если известно общее количество строк кода в системе (которое легко получить), с высокой точностью можно предсказать количество точек ветвления. Поэтому нет смысла учитывать сложную характеристику, поскольку на ее получение тратятся значительные усилия, а в результате не получается ничего, что нельзя было узнать с помощью более простой характеристики.

Характеристики, которые невозможно дифференцировать на практике

Некоторые характеристики хорошо выглядят в теории, но на практике все системы оценки оказываются примерно одинаковыми. Поэтому нет смысла включать их в модели оценки, поскольку полученные результаты будут неотличимы.

Чего не будет в этой книге

Эта книга использует язык С# (и только С#) для демонстрации и пояснения предлагаемых рекомендаций. Но она не является учебником по языку С#. Предполагается, что читатель, по крайней мере, умеет читать код на С# и знаком с программным интерфейсом его стандартных библиотек. Мы постарались сделать примеры кода простыми, используя только базовые особенности этого языка.

Кроме того, эта книга не рассказывает об идиомах С#, то есть о характерных для С# приемах выражения функциональности в коде. Мы не считаем, что можно обеспечить обслуживаемость кода с помощью определяемых конкретным языком идиом. Наоборот, представленные здесь рекомендации в значительной степени не зависят от языка и, следовательно, не зависят от языковых идиом.

Несмотря на то что книга знакомит с некоторыми шаблонами рефакторинга кода, ее не следует рассматривать как всеобъемлющий каталог таких шаблонов. Существуют книги и сайты, которые с большим основанием могут претендовать на роль таких каталогов. Эта книга посвящена описанию *причин* и *процесса* применения ряда отобранных

шаблонов рефакторинга для улучшения обслуживаемости кода. То есть эта книга служит лишь ступенькой на пути к таким каталогам.

Какую книгу прочесть следующей

Нам известно, что отдельные разработчики не контролируют всех аспектов процесса разработки. Выбор средств разработки, организация контроля качества, порядок развертывания и так далее – все эти аспекты являются важными факторами, влияющими на качество программного обеспечения, но об этом должна заботиться команда разработчиков. Поэтому эти вопросы выходят за рамки данной книги. Они будут освещены в нашей следующей книге «Building Software Teams». В ней мы опишем передовой опыт в этой области и способы оценки его применения на практике.

О компании Software Improvement Group

Хотя на обложке книги и указано только имя одного автора, в действительности авторов у этой книги гораздо больше. Настоящим автором является SIG, консалтинговая компания по управлению программным обеспечением. То есть в этой книге объединены коллективный опыт и знания консультантов SIG, которые занимаются оценкой качества программного обеспечения и консультированием в этой области с 2000 года. Мы создали уникальную сертифицированную¹ лабораторию анализа программного обеспечения, выполняющую стандартизированные проверки программного обеспечения на соответствие международным стандартам качества ISO 25010.

Одной из служб компании SIG является наша служба мониторинга рисков программного обеспечения. Клиенты службы загружают исходный код через регулярные промежутки времени (обычно раз в неделю). Этот код автоматически проверяется в лаборатории анализа программного обеспечения. Все необычное, обнаруженное в процессе автоматического анализа, исследуется консультантами SIG и обсуждается с клиентами. На момент написания книги компания SIG проанализировала в общей сложности 7,1 миллиарда строк кода, кроме того, 72,7 миллиона новых строк кода выгружается в SIG еженедельно.

Компания SIG была создана в 2000 году. Ее корни можно проследить до Голландского национального научно-исследовательского института математики и информатики (Centrum voor Wiskunde en In-

¹ Имеется в виду сертификация ISO/IEC 17025.

formatica [CWI] на голландском языке). Даже по прошествии 15 лет мы поддерживаем и ценим наши связи с научным сообществом академической разработки программного обеспечения. Консультанты SIG регулярно вносят свой вклад в научные публикации, защитили несколько кандидатских диссертаций, основанных на исследованиях в области разработки и совершенствования модели качества SIG.

Об этой версии книги

Эта версия книги основана на языке C#. Все примеры кода написаны на C# (и только на C#), в тексте книги часто встречаются названия инструментов и термины, широко используемые только в сообществе C#-разработчиков. Предполагается, что читатель имеет опыт программирования на языке C#. Однако, как уже упоминалось ранее, рекомендации, представленные в данной версии книги, не зависят от используемого языка программирования. Версия для языка Java публикуется издательством O'Reilly (и будет издана издательством «ДМК Пресс») одновременно с этой книгой.

Рекомендуемые книги

Здесь представлено 10 базовых рекомендаций для достижения высоко-го уровня обслуживаемости. Даже если это первая прочитанная вами книга, посвященная обслуживаемости, мы надеемся, что она не станет последней. Рекомендуем несколько книг для дальнейшего чтения:

«*Building Software Teams*» om *Software Improvement Group*

Дополнение к текущей книге, написанное теми же авторами. В то время как данная книга содержит рекомендации разработчикам по созданию обслуживаемого программного продукта, следующая книга освещает продвинутый процесс разработки и его оценку, основанную на подходе «цель – проблема – характеристика». Книга «*Building Software Teams*» планируется к публикации в 2016 году.

«*Refactoring: Improving the Design of Existing Code*», автор *Мартин Фаулер (Martin Fowler)*¹

Эта книга посвящена способам улучшения обслуживаемости (и других качественных характеристик) существующего кода.

¹ Фаулер М., Бек К., Брант Дж., Робертс Д., Андайк У. Рефакторинг: улучшение существующего кода. СПб.: Символ-Плюс, 2009. ISBN: 5-93286-045-6. – Прим. ред.

«*Clean Code: A Handbook of Agile Software Craftsmanship*», автор Роберт С. Мартин (Robert C. Martin) (известный также как Uncle Bob)¹

Как и данная книга, книга «Clean Code» посвящена написанию высококачественного исходного кода. Содержит более абстрактные рекомендации.

«*Code Quality: The Open Source Perspective*», автор Диомидис Спинеллис (Diomidis Spinellis)²

Подобно этой книге, «Code Quality» предоставляет рекомендации по улучшению качества кода, но, как и в книге «Clean Code», они более абстрактны.

«*Design Patterns: Elements of Reusable Object-Oriented Software*», авторы Эрик Гамма (Erich Gamma), Ричард Хелм (Richard Helm), Ральф Джонсон (Ralph Johnson) и Джон Влассидес (John Vlissides) (эта группа авторов известна также как «Банда четырех») ³

Рекомендуем прочесть эту книгу тем разработчикам программного обеспечения, которые хотят стать отличными программными архитекторами.

Соглашения, принятые в этой книге

В этой книге приняты следующие типографские соглашения:

Курсив

Используется для обозначения новых терминов, URL-адресов, адресов электронной почты, имен файлов и расширений имен файлов.

Моноширинный

Применяется для оформления листингов программ и программных элементов в тексте, таких как имена переменных, функций, баз данных, типов данных переменных окружения, операторов и ключевых слов.

¹ Мартин Р. Чистый код: создание, анализ и рефакторинг. Библиотека программиста. СПб.: Питер, 2013. ISBN: 978-5-496-00487-9. – Прим. ред.

² Спинеллис Д. Анализ программного кода на примере проектов Open Source. М.: Вильямс, 2014. ISBN: 5-8459-0604-0. – Прим. ред.

³ Гамма Э., Хелм Р., Джонсон Р., Влассидес Дж. Приемы объектно-ориентированного проектирования. СПб.: Питер, 2011. ISBN: 9785937000231. – Прим. ред.



Так обозначаются советы или рекомендации.



Так обозначаются общие примечания.



Этим значком отмечаются важные замечания

Обобщенные названия элементов ИСХОДНОГО КОДА

Для иллюстрации рекомендаций по улучшению обслуживаемости в книге используется язык C#, но сами рекомендации не являются характерными исключительно для C#. Они основываются на модели обслуживаемости SIG, не зависящей от используемой технологии и применяемой для примерно ста языков программирования и связанных с ними технологий (таких как JSP).

Языки программирования отличаются своими возможностями и (естественно) синтаксисом. Например, если в языке Java для обозначения константы используется ключевое слово `final`, в C# применяется `readonly`, имеющее тот же смысл. Другой пример: язык C# поддерживает так называемые *разделяемые классы* (partial classes)¹, которые в языке Java (в настоящее время) не поддерживаются.

Помимо различий в синтаксисе, для разных языков программирования в учебниках, пособиях и спецификациях используются разные термины. Например, понятие группы строк кода, которые выполняются как единое целое, присутствует практически в каждом языке программирования. В языках Java и C# это понятие называют *методом*. В языке Visual Basic – *подпрограммой*. В языках JavaScript и C оно известно как *функция*. В языке Pascal – это *процедура*.

Именно поэтому независимая от технологий модель нуждается в обобщенном названии для понятия группы. В табл. П.1 приведены обобщенные названия, используемые в книге.

Следующий список поясняет связи между групповыми конструкциями в табл. П.1 и практическими аспектами программирования на языке C#:

¹ <https://msdn.microsoft.com/ru-ru/library/wa80x488.aspx>. – Прим. ред.

Таблица П.1 ❖ Обобщенные понятия групп и их представление в С#

Обобщенное название	Общее определение	В языке С#
Блок кода	Наименьшая группа строк, которая может выполняться независимо	Метод или конструктор
Модуль	Наименьшая группа блоков кода	Класс верхнего уровня, интерфейс или перечисление
Компонент	Высокоуровневая часть системы, определяемая в архитектуре программного обеспечения	(Определение в языке отсутствует)
Система	Вся исследуемая кодовая база	(Определение в языке отсутствует)

От малого к большому

Понятия групп в табл. П.1 упорядочены от меньших к большим. Блок кода состоит из операторов, но сам оператор не является группирующей конструкцией.



В языке С#, как и во многих других, существует сложная взаимосвязь между операторами и содержащими их строками в файле .cs. Строка может содержать несколько операторов, но и оператор может располагаться в нескольких строках. Здесь применяется понятие строки кода (Line of Code, LoC), то есть любой строки исходного кода, заканчивающейся символами CR/LF, но не пустой и не содержащей только комментариев.

Некоторые понятия могут быть не определены в конкретном языке

Как видно из табл. П.1, некоторые из обобщенных понятий отсутствуют в языке С#. Например, в С# не существует синтаксиса для описания границ системы. Это понятие присутствует только как обобщенный термин, поскольку оно нам будет необходимо. Это не является недостатком синтаксиса С#, поскольку на практике эти границы определяются другими способами.

Некоторые из общеупотребительных терминов не используются

Возможно, вас удивило, что табл. П.1 не содержит таких популярных терминов, как субкомпоненты и подсистемы. Причина проста – они не используются в рекомендациях.

Представлены не все группирующие конструкции языка

В С# имеется больше группирующих конструкций, часть из которых отсутствует в табл. П.1. Например, для группировки классов и интерфейсов С# поддерживает пространства имен. Он также поддерживает вложенные классы. Они не перечислены в табл. П.1,

поскольку не применяются в рекомендациях. Например, нет необходимости разделять обычные и вложенные классы в формулировке рекомендаций, касающихся взаимодействий.



Пространство имен в C# нельзя отождествлять с обобщенным термином «компонент». В малых C#-системах компоненты и пространства имен могут иметь однозначное соответствие. Но в больших C#-системах пространств имен, как правило, гораздо больше, чем компонентов.

Инструменты сборки никак не отражены в общей терминологии

При разработке на языке C# интегрированная среда Visual Studio предоставляет дополнительное понятие группы: цели сборки. Тем не менее цели сборки не играют никакой роли в рекомендациях. В некоторых ситуациях можно однозначно соотнести компоненты с целями сборки, но это не является обязательным правилом.

Компоненты определяются архитектурой системы

Компоненты не являются понятием языка C#. Компоненты – это не пространства имен, не цели сборки и не решения в Visual Studio. Компоненты – это высокоуровневые строительные блоки, определяемые архитектурой программной системы. Они соответствуют блокам в блок-схеме системы. Более подробно понятие «компонент» будет описано и проиллюстрировано примерами в главе 7.

Получение примеров кода

Сопроводительные материалы к книге (примеры кода, упражнения и т. д.) можно загрузить на странице https://github.com/oreillymedia/building_maintainable_software.

Данная книга призвана оказать вам помощь в решении ваших задач. Вы можете свободно использовать примеры программного кода из этой книги в своих приложениях и в документации. Вам не нужно обращаться в издательство за разрешением, если вы не собираетесь воспроизводить существенные части программного кода. Например, если вы разрабатываете программу и используете в ней несколько отрывков программного кода из книги, вам не нужно обращаться за разрешением. Однако в случае продажи или распространения компакт-дисков с примерами из этой книги вам необходимо получить разрешение от издательства O'Reilly. Если вы отвечаете на вопросы, цитируя данную книгу или примеры из нее, получение разрешения не требуется. Но при включении существенных объемов программного

кода примеров из этой книги в вашу документацию необходимо получить разрешение издательства.

Мы приветствуем, но не требуем добавлять ссылку на первоисточник при цитировании. Под ссылкой на первоисточник мы подразумеваем указание авторов, издательства и ISBN. Например: «*Building Maintainable Software: Ten Guidelines for Future-Proof Code by Joost Visser. Copyright 2016 Software Improvement Group B.V., 978-1-4919-5352-5*».

За получением разрешения на использование значительных объемов программного кода примеров из этой книги обращайтесь по адресу permissions@oreilly.com.

Электронная библиотека Safari® Books Online

 Safari® Books Online – это виртуальная библиотека, содержащая авторитетную информацию в виде книг и видеоматериалов, созданных ведущими специалистами в области технологий и бизнеса.

Профессионалы в области технологии, разработчики программного обеспечения, веб-дизайнеры, а также бизнесмены и творческие работники используют Safari Books Online как основной источник информации для проведения исследований, решения проблем, обучения и подготовки к сертификационным испытаниям.

Библиотека Safari Books Online предлагает широкий выбор продуктов и тарифов для организаций, правительственных и учебных учреждений, а также физических лиц.

Подписчики имеют доступ к поисковой базе данных, содержащей информацию о тысячах книг, видеоматериалов и рукописей от таких издателей, как O'Reilly Media, Prentice Hall Professional, Addison-Wesley Professional, Microsoft Press, Sams, Que, Peachpit Press, Focal Press, Cisco Press, John Wiley & Sons, Syngress, Morgan Kaufmann, IBM Redbooks, Packt, Adobe Press, FT Press, Apress, Manning, New Riders, McGraw-Hill, Jones & Bartlett, Course Technology, и десятков других. За подробной информацией о Safari Books Online обращайтесь по адресу <http://www.safaribooksonline.com/>.

Как связаться с нами

С вопросами и предложениями, касающимися этой книги, обращайтесь в издательство:

Конец ознакомительного фрагмента.
Приобрести книгу можно
в интернет-магазине
«Электронный универс»
e-Univers.ru