«Великолепное руководство по стилю программирования и конструированию ПО».

Мартин Фаулер, автор книги «Refactoring»

«Книга Стива Макконнелла... это быстрый путь к мудрому программированию... Его книги увлекательны, и вы никогда не забудете то, что он рассказывает, опираясь на свой с трудом полученный опыт».

Джон Бентли, автор книги «Programming Pearls, 2d ed»

«Это просто самая лучшая книга по конструированию ПО из всех, что когда-либо попадались мне в руки. Каждый разработчик должен иметь ее и перечитывать от корки до корки каждый год. Я ежегодно перечитываю ее на протяжении вот уже девяти лет и все еще узнаю много нового!»

Джон Роббинс, автор книги «Debugging Applications for Microsoft .NET and Microsoft Windows»

«Современное ПО *должно* быть надежным и гибким, а создание защищенного кода начинается с дисциплинированного конструирования программы. За десять лет так и не появилось лучшего руководства по этой теме, чем эта книга.

Майкл Ховард, специалист по защите ПО, корпорация Microsoft; один из авторов книги «Writing Secure Code»

«Это исчерпывающее исследование тактических аспектов создания хорошо спроектированных программ. Книга Макконнелла охватывает такие разные темы, как архитектура, стандарты кодирования, тестирование, интеграция и суть разработки ПО».

Гради Буч, автор книги «Object Solutions»

«Авторитетная энциклопедия для разработчиков ПО — вот что такое „Совершенный код“. Подзаголовок „Практическое руководство по конструированию ПО“ характеризует эту 850-страничную книгу абсолютно точно. Как утверждает автор, она призвана сократить разрыв между знаниями „гуру и лучших специалистов отрасли“ (например, Йордона и Прессмана) и общепринятыми методиками разработки коммерческого ПО, а также „помочь создавать более качественные программы за меньшее время с меньшей головной болью“... Эту книгу следует иметь каждому разработчику. Ее стиль и содержание в высшей степени практичны».

Крис Лузли, автор книги «High-Performance Client/Server»

«Полная плодотворных идей книга Макконнелла „Совершенный код“ — это одна из самых понятных работ, посвященных подробному обсуждению методик разработки ПО...»

Эрик Бетке, автор книги «Game Development and Production»

«Кладезь полезной информации и рекомендаций по общим вопросам проектирования и разработки хорошего ПО».

Джон Демпстер, автор книги «The Laboratory Computer: A Practical Guide for Physiologists and Neuroscientists»

«Если вы действительно хотите улучшить навыки программирования, обязательно прочтите книгу „Совершенный код“ Стива Макконнелла».

Джин Дж. Лаброссе, автор книги «Embedded Systems Building Blocks: Complete and Ready-To-Use Modules in C»

«Стив Макконнелл написал одну из лучших книг по разработке ПО, не привязанных к конкретной среде...»

Кеннет Розен, один из авторов книги «Unix: The Complete Reference»

«Пару раз в поколение или около того появляются книги, обобщающие накопленный опыт и избавляющие вас от многих лет мучений... Не могу найти слов, чтобы адекватно описать все великолепие этой книги. „Совершенный код“ — довольно жалкое название для такой превосходной работы».

Джефф Дантеманн, журнал «PC Techniques»

«Издательство Microsoft Press опубликовало то, что я считаю самой лучшей книгой по конструированию ПО. Эта книга должна занять место на книжной полке каждого программиста».

Уоррен Keуффель, журнал «Software Development»

«Эту выдающуюся книгу следует прочесть каждому программисту».

Т. Л. (Фрэнк) Паппас, журнал «Computer»

«Если вы собираетесь стать профессиональным программистом, покупка этой книги, пожалуй, станет самым мудрым вложением средств. Можете не читать этот обзор дальше — просто идите в магазин и купите ее. Как пишет сам Макконнелл, его целью было сокращение разрыва между знаниями гуру и общепринятыми методиками разработки коммерческого ПО... Удивительно, но ему это удалось».

Ричард Матеосян, журнал «IEEE Micro»

«„Совершенный код“ — обязательное чтение для всех... кто имеет отношение к разработке ПО».

Томми Ашер, журнал «C Users Journal»

«Я вынужден сделать чуть более категоричное заявление, чем обычно, и рекомендовать книгу Стива Макконнелла „Совершенный код“ всем разработчикам без всяких оговорок... Если раньше во время работы я держал ближе всего к клавиатуре руководства по API, то теперь их место заняла книга Макконнелла».

Джим Кайл, журнал «Windows Tech Journal»

«Это лучшая книга по разработке ПО из всех, что я читал».

Эдвард Кенворт, журнал «.EXE»

«Эта книга заслуживает статуса классической, и ее в обязательном порядке должны прочесть все разработчики и те, кто ими управляет».

Питер Райт, «Program Now»

Посвящаю эту книгу своей жене Эшли, которая не имеет особого отношения к программированию, но настолько обогащает всю мою остальную жизнь, что я не могу выразить это в словах.

Steve McConnell

CODE COMPLETE

Second Edition

Microsoft® Press

Стив Макконнелл

Совершенный КОД

МАСТЕР-КЛАСС

2010

УДК 004.45
ББК 32.973.26–018.2
М15

Макконнелл С.

М15 Совершенный код. Мастер-класс / Пер. с англ. — М. : Издательство «Русская редакция», 2010. — 896 стр. : ил.

ISBN 978-5-7502-0064-1

Более 10 лет первое издание этой книги считалось одним из лучших практических руководств по программированию. Сейчас эта книга полностью обновлена с учетом современных тенденций и технологий и дополнена сотнями новых примеров, иллюстрирующих искусство и науку программирования. Опираясь на академические исследования, с одной стороны, и практический опыт коммерческих разработок ПО — с другой, автор синтезировал из самых эффективных методик и наиболее эффективных принципов ясное прагматичное руководство. Каков бы ни был ваш профессиональный уровень, с какими бы средствами разработками вы ни работали, какова бы ни была сложность вашего проекта, в этой книге вы найдете нужную информацию, она заставит вас размышлять и поможет создать совершенный код.

Книга состоит из 35 глав, предметного указателя и библиографии.

УДК 004.45
ББК 32.973.26–018.2

Подготовлено к печати по лицензионному договору с Microsoft Corporation, Редмонд, Вашингтон, США.

Microsoft, Microsoft Press, PowerPoint, Visual Basic, Windows, и Windows NT являются товарными знаками или охраняемыми товарными знаками корпорации Microsoft в США и/или других странах. Все другие товарные знаки являются собственностью соответствующих фирм.

Все названия компаний, организаций и продуктов, а также имена лиц, используемые в примерах, вымышлены и не имеют никакого отношения к реальным компаниям, организациям, продуктам и лицам.

ISBN 0-7356-1967-8 (англ.)
ISBN 978-5-7502-0064-1

© Оригинальное издание на английском языке, Steven C. McConnell, 2004

© Перевод на русский язык, Microsoft Corporation, 2004

© Оформление и подготовка к изданию, издательство «Русская редакция», 2005, 2007, 2010

Содержание

Предисловие ... **XIII**
Благодарности ... **XIX**
Контрольные списки ... **XXI**

Часть I Основы разработки ПО

1 Добро пожаловать в мир конструирования ПО! ... **2**
1.1. Что такое конструирование ПО? ... 2
1.2. Почему конструирование ПО так важно? ... 5
1.3. Как читать эту книгу ... 6

2 Метафоры, позволяющие лучше понять разработку ПО ... **8**
2.1. Важность метафор ... 8
2.2. Как использовать метафоры? ... 10
2.3. Популярные метафоры, характеризующие разработку ПО ... 12

3 Семь раз отмерь, один раз отрежь: предварительные условия ... **21**
3.1. Важность выполнения предварительных условий ... 22
3.2. Определите тип ПО, над которым вы работаете ... 28
3.3. Предварительные условия, связанные с определением проблемы ... 34
3.4. Предварительные условия, связанные с выработкой требований ... 36
3.5. Предварительные условия, связанные с разработкой архитектуры ... 41
3.6. Сколько времени следует посвятить выполнению предварительных условий? ... 52

4 Основные решения, которые приходится принимать при конструировании ... **58**
4.1. Выбор языка программирования ... 59
4.2. Конвенции программирования ... 63
4.3. Волны развития технологий ... 64
4.4. Выбор основных методик конструирования ... 66

Часть II Высококачественный код

5 Проектирование при конструировании ... **70**
5.1. Проблемы, связанные с проектированием ПО ... 71
5.2. Основные концепции проектирования ... 74
5.3. Компоненты проектирования: эвристические принципы ... 84
5.4. Методики проектирования ... 107
5.5. Комментарии по поводу популярных методологий ... 115

6 Классы ... **121**
6.1. Основы классов: абстрактные типы данных ... 122
6.2. Качественные интерфейсы классов ... 129
6.3. Вопросы проектирования и реализации ... 139

6.4. Разумные причины создания классов ... 148
6.5. Аспекты, специфические для языков ... 152
6.6. Следующий уровень: пакеты классов ... 153

7 Высококачественные методы ... 157
7.1. Разумные причины создания методов ... 160
7.2. Проектирование на уровне методов ... 163
7.3. Удачные имена методов ... 167
7.4. Насколько объемным может быть метод? ... 169
7.5. Советы по использованию параметров методов ... 170
7.6. Отдельные соображения по использованию функций ... 177
7.7. Методы-макросы и встраиваемые методы ... 178

8 Защитное программирование ... 182
8.1. Защита программы от неправильных входных данных ... 183
8.2. Утверждения ... 184
8.3. Способы обработки ошибок ... 189
8.4. Исключения ... 193
8.5. Изоляция повреждений, вызванных ошибками ... 198
8.6. Отладочные средства ... 200
8.7. Доля защитного программирования в промышленной версии ... 204
8.8. Защита от защитного программирования ... 206

9 Процесс программирования с псевдокодом ... 209
9.1. Этапы создания классов и методов ... 210
9.2. Псевдокод для профи ... 211
9.3. Конструирование методов с использованием ППП ... 214
9.4. Альтернативы ППП ... 225

Часть III Переменные

10 Общие принципы использования переменных ... 230
10.1. Что вы знаете о данных? ... 231
10.2. Грамотное объявление переменных ... 232
10.3. Принципы инициализации переменных ... 233
10.4. Область видимости ... 238
10.5. Персистентность ... 245
10.6. Время связывания ... 246
10.7. Связь между типами данных и управляющими структурами ... 247
10.8. Единственность цели каждой переменной ... 249

11 Сила имен переменных ... 252
11.1. Общие принципы выбора имен переменных ... 253
11.2. Именование конкретных типов данных ... 257
11.3. Сила конвенций именования ... 263
11.4. Неформальные конвенции именования ... 264
11.5. Стандартизованные префиксы ... 272
11.6. Грамотное сокращение имен переменных ... 274
11.7. Имена, которых следует избегать ... 277

12 Основные типы данных ... 282
12.1. Числа в общем ... 283
12.2. Целые числа ... 284
12.3. Числа с плавающей запятой ... 286
12.4. Символы и строки ... 289
12.5. Логические переменные ... 292
12.6. Перечислимые типы ... 294
12.7. Именованные константы ... 299
12.8. Массивы ... 301
12.9. Создание собственных типов данных (псевдонимы) ... 303
13 Нестандартные типы данных ... 310
13.1. Структуры ... 310
13.2. Указатели ... 314
13.3. Глобальные данные ... 326

Часть IV Операторы

14 Организация последовательного кода ... 338
14.1. Операторы, следующие в определенном порядке ... 338
14.2. Операторы, следующие в произвольном порядке ... 342
15 Условные операторы ... 346
15.1. Операторы if ... 346
15.2. Операторы case ... 353
16 Циклы ... 359
16.1. Выбор типа цикла ... 359
16.2. Управление циклом ... 365
16.3. Простое создание цикла — изнутри наружу ... 378
16.4. Соответствие между циклами и массивами ... 379
17 Нестандартные управляющие структуры ... 382
17.1. Множественные возвраты из метода ... 382
17.2. Рекурсия ... 385
17.3. Оператор goto ... 389
17.4. Перспективы нестандартных управляющих структур ... 401
18 Табличные методы ... 404
18.1. Основные вопросы применения табличных методов ... 405
18.2. Таблицы с прямым доступом ... 406
18.3. Таблицы с индексированным доступом ... 418
18.4. Таблицы со ступенчатым доступом ... 419
18.5. Другие примеры табличного поиска ... 422
19 Общие вопросы управления ... 424
19.1. Логические выражения ... 424
19.2. Составные операторы (блоки) ... 436
19.3. Пустые выражения ... 437
19.4. Укрощение опасно глубокой вложенности ... 438

19.5. Основа программирования: структурное программирование 448
19.6. Управляющие структуры и сложность 450

Часть V Усовершенствование кода

20 Качество ПО 456
20.1. Характеристики качества ПО 456
20.2. Методики повышения качества ПО 459
20.3. Относительная эффективность методик контроля качества ПО 462
20.4. Когда выполнять контроль качества ПО? 466
20.5. Главный Закон Контроля Качества ПО 467

21 Совместное конструирование 471
21.1. Обзор методик совместной разработки ПО 472
21.2. Парное программирование 475
21.3. Формальные инспекции 477
21.4. Другие методики совместной разработки ПО 484
21.5. Сравнение методик совместного конструирования 487

22 Тестирование, выполняемое разработчиками 490
22.1. Тестирование, выполняемое разработчиками, и качество ПО 492
22.2. Рекомендуемый подход к тестированию, выполняемому разработчиками 494
22.3. Приемы тестирования 496
22.4. Типичные ошибки 507
22.5. Инструменты тестирования 513
22.6. Оптимизация процесса тестирования 518
22.7. Протоколы тестирования 520

23 Отладка 524
23.1. Общие вопросы отладки 524
23.2. Поиск дефекта 529
23.3. Устранение дефекта 539
23.4. Психологические аспекты отладки 543
23.5. Инструменты отладки — очевидные и не очень 545

24 Рефакторинг 551
24.1. Виды эволюции ПО 552
24.2. Введение в рефакторинг 553
24.3. Отдельные виды рефакторинга 559
24.4. Безопасный рефакторинг 566
24.5. Стратегии рефакторинга 568

25 Стратегии оптимизации кода 572
25.1. Общее обсуждение производительности ПО 573
25.2. Введение в оптимизацию кода 576
25.3. Где искать жир и патоку? 583
25.4. Оценка производительности 588

25.5. Итерация ... 590
25.6. Подход к оптимизации кода: резюме ... 591

26 Методики оптимизации кода ... 595
26.1. Логика ... 596
26.2. Циклы ... 602
26.3. Изменения типов данных ... 611
26.4. Выражения ... 616
26.5. Методы ... 625
26.6. Переписывание кода на низкоуровневом языке ... 626
26.7. Если что-то одно изменяется, что-то другое всегда остается постоянным ... 629

Часть VI Системные вопросы

27 Как размер программы влияет на конструирование ... 634
27.1. Взаимодействие и размер ... 635
27.2. Диапазон размеров проектов ... 636
27.3. Влияние размера проекта на возникновение ошибок ... 636
27.4. Влияние размера проекта на производительность ... 638
27.5. Влияние размера проекта на процесс разработки ... 639

28 Управление конструированием ... 645
28.1. Поощрение хорошего кодирования ... 646
28.2. Управление конфигурацией ... 649
28.3. Оценка графика конструирования ... 655
28.4. Измерения ... 661
28.5. Гуманное отношение к программистам ... 664
28.6. Управление менеджером ... 670

29 Интеграция ... 673
29.1. Важность выбора подхода к интеграции ... 673
29.2. Частота интеграции — поэтапная или инкрементная? ... 675
29.3. Стратегии инкрементной интеграции ... 678
29.4. Ежедневная сборка и дымовые тесты ... 686

30 Инструменты программирования ... 694
30.1. Инструменты для проектирования ... 695
30.2. Инструменты для работы с исходным кодом ... 695
30.3. Инструменты для работы с исполняемым кодом ... 700
30.4. Инструменты и среды ... 704
30.5. Создание собственного программного инструментария ... 705
30.6. Волшебная страна инструментальных средств ... 707

Часть VII Мастерство программирования

31 Форматирование и стиль ... 712
31.1. Основные принципы форматирования ... 713
31.2. Способы форматирования ... 720
31.3. Стили форматирования ... 721

31.4. Форматирование управляющих структур ... 728
31.5. Форматирование отдельных операторов ... 736
31.6. Размещение комментариев ... 747
31.7. Размещение методов ... 750
31.8. Форматирование классов ... 752

32 Самодокументирующийся код ... 760

32.1. Внешняя документация ... 760
32.2. Стиль программирования как вид документации ... 761
32.3. Комментировать или не комментировать? ... 764
32.4. Советы по эффективному комментированию ... 768
32.5. Методики комментирования ... 774
32.6. Стандарты IEEE ... 795

33 Личность ... 800

33.1. Причем тут характер? ... 801
33.2. Интеллект и скромность ... 802
33.3. Любопытство ... 803
33.4. Профессиональная честность ... 806
33.5. Общение и сотрудничество ... 809
33.6. Творчество и дисциплина ... 809
33.7. Лень ... 810
33.8. Свойства, которые менее важны, чем кажется ... 811
33.9. Привычки ... 813

34 Основы мастерства ... 817

34.1. Боритесь со сложностью ... 817
34.2. Анализируйте процесс разработки ... 819
34.3. Пишите программы в первую очередь для людей и лишь во вторую — для компьютеров ... 821
34.4. Программируйте с использованием языка, а не на языке ... 823
34.5. Концентрируйте внимание с помощью соглашений ... 824
34.6. Программируйте в терминах проблемной области ... 825
34.7. Опасайтесь падающих камней ... 827
34.8. Итерируйте, итерируйте и итерируйте ... 830
34.9. И да отделена будет религия от разработки ПО ... 831

35 Где искать дополнительную информацию ... 834

35.1. Информация о конструировании ПО ... 835
35.2. Не связанные с конструированием темы ... 836
35.3. Периодические издания ... 838
35.4. Список литературы для разработчика ПО ... 839
35.5. Профессиональные ассоциации ... 841

Библиография ... 842

Предметный указатель ... 863

Об авторе ... 868

Предисловие

> *Разрыв между самыми лучшими и средними методиками разработки ПО очень широк — вероятно, шире, чем в любой другой инженерной дисциплине. Средство распространения информации о хороших методиках сыграло бы весьма важную роль.*
>
> *Фред Брукс (Fred Brooks)*

Моей главной целью при написании этой книги было сокращение разрыва между знаниями гуру и лучших специалистов отрасли, с одной стороны, и общепринятыми методиками разработки коммерческого ПО — с другой. Многие эффективные методики программирования годами скрываются в журналах и научных работах, прежде чем становятся доступными программистской общественности.

Хотя передовые методики разработки ПО в последние годы быстро развивались, общепринятые практически стояли на месте. Многие программы все еще полны ошибок, поставляются с опозданием и не укладываются в бюджет, а многие не отвечают требованиям пользователей. Ученые обнаружили эффективные методики, устраняющие большинство проблем, которые отравляют нашу жизнь с 1970-х годов. Однако из-за того, что эти методики редко покидают страницы узкоспециализированных технических изданий, в большинстве компаний по разработке ПО они еще не используются. Установлено, что для широкого распространения исследовательских разработок обычно требуется от 5 до 15 и более лет (Raghavan and Chand, 1989; Rogers, 1995; Parnas, 1999). Данная книга призвана ускорить этот процесс и сделать важные открытия доступными средним программистам.

Кому следует прочитать эту книгу?

Исследования и опыт программирования, отраженные в этой книге, помогут вам создавать высококачественное ПО и выполнять свою работу быстрее и эффективнее. Прочитав ее, вы поймете, почему вы сталкивались с проблемами в прошлом, и узнаете, как избегать их в будущем. Описанные мной методики программирования помогут вам сохранять контроль над крупными проектами, а также успешно сопровождать и изменять ПО при изменении требований.

Опытные программисты

Эта книга окажется полезной опытным программистам, желающим получить всестороннее и удобное руководство по разработке ПО. Так как эта книга фокусируется на конструировании — самой известной части жизненного цикла ПО, — описанные в ней методики будут понятны и программистам, имеющим соответствующее образование, и программистам-самоучкам.

Технические лидеры

Многие технические лидеры используют первое издание этой книги для обучения менее опытных членов своих групп. Вы также можете использовать эту книгу для восполнения пробелов в своих знаниях. Если вы — опытный программист, то, наверное, согласитесь не со всеми моими выводами (обратное было бы странным), но, если вы прочитаете весь материал и обдумаете каждый поднятый вопрос, едва ли какая-то возникшая проблема конструирования окажется для вас новой.

Программисты-самоучки

Если вы не имеете специального образования, вы не одиноки. Ежегодно программистами становятся около 50 000 человек (BLS, 2004, Hecker 2004), однако число дипломов, вручаемых ежегодно в нашей отрасли, составляет лишь около 35 000 (NCES, 2002). Легко прийти к выводу, что многие программисты изучают разработку ПО самостоятельно. Программисты-самоучки встречаются среди инженеров, бухгалтеров, ученых, преподавателей, владельцев малого бизнеса и представителей других профессий, которые занимаются программированием в рамках своей работы, но не всегда считают себя программистами. Каким бы ни было ваше программистское образование, в этом руководстве вы найдете информацию об эффективных методиках программирования.

Студенты

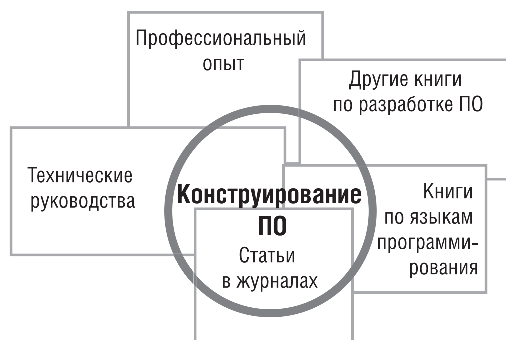
В отличие от программистов, которые обладают опытом, но не могут похвастаться специальным обучением, недавние выпускники вузов часто имеют обширные теоретические знания, но плохо владеют практическими ноу-хау, связанными с созданием реальных программ. Передача практических навыков хорошего кодирования зачастую идет медленно, в ритуальных танцах архитекторов ПО, лидеров проектов, аналитиков и более опытных программистов. Еще чаще эти навыки приобретаются программистами в результате собственных проб и ошибок. Эта книга — альтернатива традиционным неспешным интеллектуальным ритуалам. В ней собраны полезные советы и эффективные стратегии разработки, которые ранее можно было узнать главным образом только непосредственно у других людей. Это трамплин для студентов, переходящих из академической среды в профессиональную.

Где еще можно найти эту информацию?

В этой книге собраны методики конструирования из самых разнообразных источников. Многие знания о конструировании не только разрозненны, но и годами не попадают в печатные издания (Hildebrand, 1989; McConnell, 1997a). В эффективных, мощных методиках программирования, используемых лучшими программистами, нет ничего мистического, однако в повседневной череде неотложных задач очень немногие эксперты выкраивают время на то, чтобы поделиться своим опытом. Таким образом, программистам трудно найти хороший источник информации о программировании.

Методики, описанные в этой книге, заполняют пустоту, остающуюся в знаниях программистов после прочтения вводных и более серьезных учебников по программированию. Что читать человеку, изучившему книги типа «Introduction to Java», «Advanced Java» и «Advanced Advanced Java» и желающему узнать о программировании больше? Вы можете читать книги о процессорах Intel или Motorola, функциях ОС Microsoft Windows или Linux или о другом языке программирования — невозможно эффективно программировать, не имея хорошего представления о таких деталях. Но эта книга относится к числу тех немногих, в которых обсуждается программирование как таковое. Наибольшую пользу приносят те методики, которые можно использовать независимо от среды или языка. В других источниках такие методики обычно игнорируются, и именно поэтому я сосредоточился на них.

Как показано ниже, информация, представленная в этой книге, выжата из многих источников. Единственным другим способом получения этой информации является изучение горы книг и нескольких сотен технических журналов, дополненное значительным реальным опытом. Если вы уже проделали все это, данная книга все равно окажется вам полезной как удобный справочник.



Главные достоинства этой книги

Какой бы ни была ваша ситуация, эта книга поможет вам создавать более качественные программы за меньшее время с меньшей головной болью.

Полное руководство по конструированию ПО В этой книге обсуждаются такие общие аспекты конструирования, как качество ПО и подходы к размышлению о программировании. В то же время мы погрузимся в такие детали конструирования, как этапы создания классов, использование данных и управляющих структур, отладка, рефакторинг и методики и стратегии оптимизации кода. Чтобы изучить эти вопросы, вам не нужно читать книгу от корки до корки. Материал организован так, чтобы вы могли легко найти конкретную интересующую вас информацию.

Готовые к использованию контрольные списки Эта книга включает десятки контрольных списков, позволяющих оценить архитектуру программы, подход к проектированию, качество классов и методов, имена переменных, управляющие структуры, форматирование, тесты и многое другое.

Самая актуальная информация В этом руководстве вы найдете описания ряда самых современных методик, многие из которых еще не стали общепринятыми. Так как эта книга основана и на практике, и на исследованиях, рассмотренные в ней методики будут полезны еще многие годы.

Более общий взгляд на разработку ПО Эта книга даст вам шанс подняться над суетой повседневной борьбы с проблемами и узнать, что работает, а что нет. Мало кто из практикующих программистов обладает временем, необходимым для прочтения сотен книг и журнальных статей, обобщенных в этом руководстве. Исследования и реальный опыт, на которых основана данная книга, помогут вам проанализировать ваши проекты и позволят принимать стратегические решения, чтобы не приходилось бороться с теми же врагами снова и снова.

Объективность Некоторые книги по программированию содержат 1 грамм информации на 10 граммов рекламы. Здесь вы найдете сбалансированные обсуждения достоинств и недостатков каждой методики. Вы знаете свой конкретный проект лучше всех, и эта книга предоставит вам объективную информацию, нужную для принятия грамотных решений в ваших обстоятельствах.

Независимость от языка Описанные мной методики позволяют выжать максимум почти из любого языка, будь то C++, C#, Java, Microsoft Visual Basic или другой похожий язык.

Многочисленные примеры кода Эта книга содержит почти 500 примеров хорошего и плохого кода. Их так много потому, что лично я лучше всего учусь на примерах. Думаю, это относится и к другим программистам.

Примеры написаны на нескольких языках, потому что освоение более одного языка часто является поворотным пунктом в карьере профессионального программиста. Как только программист понимает, что принципы программирования не зависят от синтаксиса конкретного языка, он начинает приобретать знания, позволяющие достичь новых высот качества и производительности труда.

Чтобы как можно более облегчить бремя применения нескольких языков, я избегал редких возможностей языков, кроме тех фрагментов, в которых именно они и обсуждаются. Вам не нужно понимать каждый нюанс фрагментов кода, чтобы понять их суть. Если вы сосредоточитесь на обсуждаемых моментах, вы сможете читать код на любом языке. Чтобы сделать вашу задачу еще легче, я пояснил важные части примеров.

Доступ к другим источникам информации В данном руководстве приводятся подробные сведения о конструировании ПО, но едва ли это последнее слово. В разделах «Дополнительные ресурсы» я указал другие книги и статьи, которые вы можете прочитать, если заинтересуетесь той или иной темой.

http://cc2e.com/1234

Web-сайт книги Обновленные контрольные списки, списки книг и журнальных статей, Web-ссылки и другую информацию можно найти на Web-сайте *cc2e.com*. Для получения информации, связанной с «Code Complete, 2d ed.», введите в браузере *cc2e.com/* и четырехзначное число, пример которого показан слева. Читая книгу, вы много раз натолкнетесь на такие ссылки.

Что побудило меня написать эту книгу?

Необходимость руководств, отражающих знания об эффективных методиках разработки ПО, ясна всем членам сообщества разработчиков. Согласно отчету совета Computer Science and Technology Board максимальное повышение качества и продуктивности разработки ПО будет достигнуто благодаря систематизации, унификации и распространению существующих знаний об эффективных методиках разработки (CSTB, 1990; McConnell, 1997a). Совет пришел к выводу, что стратегия распространения этих знаний должна быть основана на концепции руководств по разработке ПО.

Тема конструирования игнорировалась

Одно время разработка ПО и кодирование рассматривались как одно и то же. Однако по мере идентификации разных процессов цикла разработки ПО лучшие умы отрасли стали посвящать время анализу и обсуждению методик управления проектами, выработки требований, проектирования и тестирования. Из-за пристального внимания к этим новым областям конструирование кода превратилось в бедного родственника разработки ПО.

Кроме того, обсуждению конструирования препятствовало предположение, согласно которому подход к конструированию как к отдельному *процессу* разработки ПО подразумевает, что конструирование нужно рассматривать при этом как отдельный *этап*. На самом деле процессы и этапы разработки не обязаны быть связаны какими-то отношениями, и обсуждение процесса конструирования полезно независимо от того, выполняются ли другие процессы разработки ПО как этапы, итерации или как-то иначе.

Конструирование важно

Другая причина того, что конструирование игнорируется учеными и авторами, заключается в ошибочной идее, что в сравнении с другими процессами разработки ПО конструирование является относительно механическим процессом, допускающим мало возможностей улучшения. Ничто не может быть дальше от истины.

На конструирование кода обычно приходятся около 65% работы в небольших и 50% в средних проектах. Во время конструирования допускаются около 75% ошибок в неболь-

ших проектах и от 50 до 75% в средних и крупных. Очевидно, что любой процесс, связанный с такой долей ошибок, можно значительно улучшить (подробнее эти статистические данные рассматриваются в главе 27).

Некоторые авторы указывают, что, хотя ошибки конструирования и составляют высокий процент от общего числа ошибок, их обычно дешевле исправлять, чем ошибки в требованиях или архитектуре, поэтому они менее важны. Утверждение, что ошибки конструирования дешевле исправлять, верно, но вводит в заблуждение, потому что стоимость неисправленной ошибки конструирования может быть крайней высокой. Ученые обнаружили, что одними из самых дорогих ошибок в истории, приведшими к убыткам в сотни миллионов долларов, были мелкие ошибки кодирования (Weinberg, 1983; SEN, 1990). Невысокая стоимость исправления ошибок не подразумевает, что их исправление можно считать низкоприоритетной задачей.

Ирония ослабления внимания к конструированию состоит в том, что конструирование — единственный процесс, который выполняется всегда. Требования можно предположить, а не разработать, архитектуру можно обрисовать в самых общих чертах, а тестирование можно сократить или вообще опустить. Но если вы собираетесь написать программу, избежать конструирования не удастся, и это делает конструирование на редкость плодотворной областью улучшения методик разработки.

Отсутствие похожих книг

Когда я начал подумывать об этой книге, я был уверен, что кто-то другой уже написал об эффективных методиках конструирования. Необходимость такой книги казалась очевидной. Но я обнаружил лишь несколько книг о конструировании, описывающих лишь некоторые его аспекты. Одни были написаны 15 или более лет назад и были основаны на относительно редких языках, таких как ALGOL, PL/I, Ratfor и Smalltalk. Другие были написаны профессорами, не работавшими над реальным кодом. Профессора писали о методиках, работающих в студенческих проектах, но часто не имели представления о том, как эти методики проявят себя в полномасштабных средах разработки. В третьих книгах авторы рекламировали новейшие методологии, игнорируя многие зрелые методики, эффективность которых прошла проверку временем.

Короче говоря, я не смог найти ни одной книги, автор которой хотя бы попытался отразить в ней практические приемы программирования, возникшие благодаря накоплению профессионального опыта, отраслевым исследованиям и академическим изысканиям. Обсуждение конструирования нужно было привести в соответствие современным языкам программирования, объектно-ориентированному программированию и ведущим методикам разработки. Ясно, что книгу о программировании должен был написать человек, знакомый с последними достижениями в области теории и в то же время создавший достаточно реального кода, чтобы хорошо представлять состояние практической сферы. Я писал эту книгу как всестороннее обсуждение конструирования кода, имеющее целью передачу знаний от одного программиста другому.

> Когда вместе собираются критики, они говорят о Теме, Композиции и Идее. Когда вместе собираются художники, они говорят о том, где купить дешевый скипидар.
>
> *Пабло Пикассо*

К читателям

Я буду рад получить от вас вопросы по темам, обсуждаемым в этой книге, сообщения об обнаруженных ошибках, комментарии и предложения. Для связи со мной используйте адрес *stevemcc@construx.com* или мой Web-сайт *www.stevemcconnell.com*.

Беллвью, штат Вашингтон
30 мая 2004 года

Служба поддержки Microsoft Learning Technical Support

Мы приложили все усилия, чтобы обеспечить точность сведений, изложенных в этой книге. Поправки к книгам издательства Microsoft Press публикуются в Интернете по адресу: *http://www.microsoft.com/learning/support/*

Чтобы подключиться к базе знаний Microsoft и задать вопрос или запросить ту или иную информацию, откройте страницу:

http://www.microsoft.com/learning/support/search.asp

Если у вас есть замечания, вопросы или предложения по поводу этой книги, присылайте их в Microsoft Press по обычной почте:

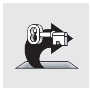
Microsoft Press
Attn: Code Complete 2E Editor
One Microsoft Way
Redmond, WA 98052-6399

или по электронной почте:

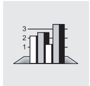
mspinput@microsoft.com

Примечание издателя перевода

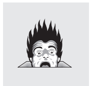
В книге приняты следующие условные графические обозначения:



Ключевой момент



Достоверные данные



Ужасный код

Благодарности

Книги никогда не создаются в одиночку (по крайней мере это относится ко всем моим книгам), а работа над вторым изданием — еще более коллективное предприятие.

Мне хотелось бы поблагодарить всех, кто принял участие в обзоре данной книги: это Хакон Агустссон (Нбkon Бgъstsson), Скотт Эмблер (Scott Ambler), Уилл Барнс (Will Barns), Уильям Д. Бартоломью (William D. Bartholomew), Ларс Бергстром (Lars Bergstrom), Ян Брокбанк (Ian Brockbank), Брюс Батлер (Bruce Butler), Джей Цинкотта (Jay Cincotta), Алан Купер (Alan Cooper), Боб Коррик (Bob Corrick), Эл Корвин (Al Corwin), Джерри Девилль (Jerry Deville), Джон Ивз (Jon Eaves), Эдвард Эстрада (Edward Estrada), Стив Гоулдстоун (Steve Gouldstone), Оуэйн Гриффитс (Owain Griffiths), Мэтью Харрис (Matthew Harris), Майкл Ховард (Michael Howard), Энди Хант (Andy Hunt), Кевин Хатчисон (Kevin Hutchison), Роб Джаспер (Rob Jasper), Стивен Дженкинс (Stephen Jenkins), Ральф Джонсон (Ralph Johnson) и его группа разработки архитектуры ПО из Иллинойского университета, Марек Конопка (Marek Konopka), Джефф Лэнгр (Jeff Langr), Энди Лестер (Andy Lester), Митика Ману (Mitica Manu), Стив Маттингли (Steve Mattingly), Гарет Маккоан (Gareth McCaughan), Роберт Макговерн (Robert McGovern), Скотт Мейерс (Scott Meyers), Гарет Морган (Gareth Morgan), Мэтт Пелокин (Matt Peloquin), Брайан Пфладж (Bryan Pflug), Джеффри Рихтер (Jeffrey Richter), Стив Ринн (Steve Rinn), Даг Розенберг (Doug Rosenberg), Брайан Сен-Пьер (Brian St. Pierre), Диомидис Спиннелис (Diomidis Spinellis), Мэтт Стивенс (Matt Stephens), Дэйв Томас (Dave Thomas), Энди Томас-Крамер (Andy Thomas-Cramer), Джон Влиссидес (John Vlissides), Павел Возенилек (Pavel Vozenilek), Денни Уиллифорд (Denny Williford), Джек Вули (Jack Woolley) и Ди Зомбор (Dee Zsombor).

Сотни читателей прислали комментарии к первому изданию этой книги, и еще больше — ко второму. Спасибо всем, кто потратил время, чтобы поделиться в той или иной форме своим мнением.

Хочу особо поблагодарить рецензентов из Construx Software, которые провели формальную инспекцию всей рукописи: это Джейсон Хиллз (Jason Hills), Брейди Хонсингер (Bradey Honsinger), Абдул Низар (Abdul Nizar), Том Рид (Tom Reed) и Памела Перро (Pamela Perrott). Я был поистине удивлен тщательностью их обзора, особенно если учесть, сколько глаз изучило эту книгу до того, как они начали работать с ней. Спасибо также Брейди, Джейсону и Памеле за помощь в создании Web-сайта *cc2e.com*.

Мне было очень приятно работать с Девон Масгрейв (Devon Musgrave) — редактором этой книги. Я работал со многими прекрасными редакторами в других проектах, но даже на их фоне Девон выделяется добросовестностью и легким

характером. Спасибо, Девон! Благодарю Линду Энглман (Linda Engleman), которая поддержала идею второго издания — без нее эта книга не появилась бы. Благодарю также других сотрудников издательства Microsoft Press, в их число входят Робин ван Стинбург (Robin Van Steenburgh), Элден Нельсон (Elden Nelson), Карл Дилтц (Carl Diltz), Джоэл Панчо (Joel Panchot), Патрисия Массерман (Patricia Masserman), Билл Майерс (Bill Myers), Сэнди Резник (Sandi Resnick), Барбара Норфлит (Barbara Norfleet), Джеймс Крамер (James Kramer) и Прескотт Классен (Prescott Klassen).

Я хочу еще раз сказать спасибо сотрудникам Microsoft Press, участвовавшим в подготовке первого издания книги: это Элис Смит (Alice Smith), Арлен Майерс (Arlene Myers), Барбара Раньян (Barbara Runyan), Кэрол Люк (Carol Luke), Конни Литтл (Connie Little), Дин Холмс (Dean Holmes), Эрик Стру (Eric Stroo), Эрин О'Коннор (Erin O'Connor), Джинни Макгиверн (Jeannie McGivern), Джефф Кэри (Jeff Carey), Дженнифер Харрис (Jennifer Harris), Дженнифер Вик (Jennifer Vick), Джудит Блох (Judith Bloch), Кэтрин Эриксон (Katherine Erickson), Ким Эгглстон (Kim Eggleston), Лиза Сэндбург (Lisa Sandburg), Лиза Теобальд (Lisa Theobald), Маргарет Харгрейв (Margarite Hargrave), Майк Халворсон (Mike Halvorson), Пэт Фоджетт (Pat Forgette), Пегги Герман (Peggy Herman), Рут Петтис (Ruth Pettis), Салли Брунсмен (Sally Brunsman), Шон Пек (Shawn Peck), Стив Мюррей (Steve Murray), Уоллис Болц (Wallis Bolz) и Заафар Хаснаин (Zaafar Hasnain).

Наконец, я хотел бы выразить благодарность рецензентам, внесшим такой большой вклад в первое издание книги: это Эл Корвин (Al Corwin), Билл Кистлер (Bill Kiestler), Брайан Догерти (Brian Daugherty), Дэйв Мур (Dave Moore), Грег Хичкок (Greg Hitchcock), Хэнк Меуре (Hank Meuret), Джек Вули (Jack Woolley), Джой Уайрик (Joey Wyrick), Марго Пейдж (Margot Page), Майк Клейн (Mike Klein), Майк Зевенберген (Mike Zevenbergen), Пэт Форман (Pat Forman), Питер Пэт (Peter Pathe), Роберт Л. Гласс (Robert L. Glass), Тэмми Форман (Tammy Forman), Тони Пискулли (Tony Pisculli) и Уэйн Бердсли (Wayne Beardsley). Особо благодарю Тони Гарланда (Tony Garland) за его обстоятельный обзор: за 12 лет я еще лучше понял, как выиграла эта книга от тысяч комментариев Тони.

Контрольные списки

Требования ... 42
Архитектура ... 54
Предварительные условия ... 59
Основные методики конструирования ... 69
Проектирование при конструировании ... 122
Качество классов ... 157
Высококачественные методы ... 185
Защитное программирование ... 211
Процесс программирования с псевдокодом ... 233
Общие вопросы использования данных ... 257
Именование переменных ... 288
Основные данные ... 316
Применение необычных типов данных ... 343
Организация последовательного кода ... 353
Использование условных операторов ... 365
Циклы ... 388
Нестандартные управляющие структуры ... 410
Табличные методы ... 429
Вопросы по управляющим структурам ... 459
План контроля качества ... 476
Эффективное парное программирование ... 484
Эффективные инспекции ... 491
Тесты ... 532
Отладка ... 559
Разумные причины выполнения рефакторинга ... 570
Виды рефакторинга ... 577
Безопасный рефакторинг ... 584
Стратегии оптимизации кода ... 607
Методики оптимизации кода ... 642
Управление конфигурацией ... 669
Интеграция ... 707
Инструменты программирования ... 724
Форматирование ... 773
Самодокументирующийся код ... 780
Хорошие методики комментирования ... 816

Часть I

ОСНОВЫ РАЗРАБОТКИ ПО

- **Глава 1.** Добро пожаловать в мир конструирования ПО!
- **Глава 2.** Метафоры, позволяющие лучше понять разработку ПО
- **Глава 3.** Семь раз отмерь, один раз отрежь: предварительные условия
- **Глава 4.** Основные решения, которые приходится принимать при конструировании

ГЛАВА 1

Добро пожаловать в мир конструирования ПО!

http://cc2e.com/0178

Содержание

- 1.1. Что такое конструирование ПО?
- 1.2. Почему конструирование ПО так важно?
- 1.3. Как читать эту книгу

Связанные темы

- Кому следует прочитать эту книгу? (см. предисловие)
- Какую выгоду можно извлечь, прочитав эту книгу? (см. предисловие)
- Что побудило меня написать эту книгу? (см. предисловие)

Значение слова «конструирование» вне контекста разработки ПО известно всем: это то, что делают строители при сооружении жилого дома, школы или небоскреба. В детстве вы наверняка собирали разные предметы из «конструктора». Вообще под «конструированием» понимают процесс создания какого-нибудь объекта. Этот процесс может включать некоторые аспекты планирования, проектирования и тестирования, но чаще всего «конструированием» называют практическую часть создания чего-либо.

1.1. Что такое конструирование ПО?

Разработка ПО — непростой процесс, который может включать множество компонентов. Вот какие составляющие разработки ПО определили ученые за последние 25 лет:

- определение проблемы;
- выработка требований;
- создание плана конструирования;
- разработка архитектуры ПО, или высокоуровневое проектирование;
- детальное проектирование;
- кодирование и отладка;

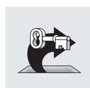
- блочное тестирование;
- интеграционное тестирование;
- интеграция;
- тестирование системы;
- корректирующее сопровождение.

Если вы работали над неформальными проектами, то можете подумать, что этот список весьма бюрократичен. Если вы работали над слишком формальными проектами, вы это *знаете*! Достичь баланса между слишком слабым и слишком сильным формализмом нелегко — об этом мы еще поговорим.

Если вы учились программировать самостоятельно или работали преимущественно над неформальными проектами, вы, возможно, многие действия по созданию продукта объединили в одну категорию «программирование». Если вы работаете над неформальными проектами, то скорее всего, думая о создании ПО, вы представляете себе тот процесс, который ученые называют «конструированием».

Такое интуитивное представление о «конструировании» довольно верно, однако оно страдает от недостатка перспективы. Поэтому конструирование целесообразно рассматривать в контексте других процессов: это помогает сосредоточиться на задачах конструирования и уделять адекватное внимание другим важным действиям, к нему не относящимся. На рис. 1-1 показано место конструирования среди других процессов разработки ПО.

Рис. 1-1. *Процессы конструирования изображены внутри серого эллипса. Главными компонентами конструирования являются кодирование и отладка, однако оно включает и детальное проектирование, блочное тестирование, интеграционное тестирование и другие процессы*



Как видите, конструирование состоит преимущественно из кодирования и отладки, однако включает и детальное проектирование, создание плана конструирования, блочное тестирование, интеграцию, интеграцион-

ное тестирование и другие процессы. Если бы эта книга была посвящена всем аспектам разработки ПО, в ней было бы приведено сбалансированное обсуждение всех процессов. Однако это руководство по методам конструирования, так что остальных тем я почти не касаюсь. Если бы эта книга была собакой, она тщательно принюхивалась бы к конструированию, виляла хвостом перед проектированием и тестированием и лаяла на прочие процессы.

Иногда конструирование называют «кодированием» или «программированием». «Кодирование» кажется мне в данном случае не лучшим термином, так как он подразумевает механическую трансляцию разработанного плана в команды языка программирования, тогда как конструирование вовсе не механический процесс и часто связано с творчеством и анализом. Смысл слов «программирование» и «конструирование» кажется мне похожим, и я буду использовать их как равноправные.

На рис. 1-1 разработка ПО была изображена в «плоском» виде; более точным отражением содержания этой книги является рис. 1-2.

Рис. 1-2. *Кодирование и отладка, детальное проектирование, создание плана конструирования, блочное тестирование, интеграция, интеграционное тестирование и другие процессы обсуждаются в данной книге примерно в такой пропорции*

На рис. 1-1 и 1-2 процессы конструирования представлены с общей точки зрения. Но что можно сказать об их деталях? Вот некоторые конкретные задачи, связанные с конструированием:

- проверка выполнения условий, необходимых для успешного конструирования;
- определение способов последующего тестирования кода;
- проектирование и написание классов и методов;
- создание и присвоение имен переменным и именованным константам;
- выбор управляющих структур и организация блоков команд;

- блочное тестирование, интеграционное тестирование и отладка собственного кода;
- взаимный обзор кода и низкоуровневых программных структур членами группы;
- «шлифовка» кода путем его тщательного форматирования и комментирования;
- интеграция программных компонентов, созданных по отдельности;
- оптимизация кода, направленная на повышение его быстродействия, и снижение степени использования ресурсов.

Еще более полное представление о процессах и задачах конструирования вы получите, просмотрев содержание книги.

Конструирование включает так много задач, что вы можете спросить: «Ладно, а что *не* является частью конструирования?» Хороший вопрос. В конструирование не входят такие важные процессы, как управление, выработка требований, разработка архитектуры приложения, проектирование пользовательского интерфейса, тестирование системы и ее сопровождение. Все они не меньше, чем конструирование, влияют на конечный успех проекта — по крайней мере любого проекта, который требует усилий более одного-двух человек и длится больше нескольких недель. Все эти процессы стали предметом хороших книг, многие из которых я указал в разделах «Дополнительные ресурсы» и в главе 35.

1.2. Почему конструирование ПО так важно?

Раз уж вы читаете эту книгу, вы наверняка понимаете важность улучшения качества ПО и повышения производительности труда разработчиков. Многие из самых удивительных современных проектов основаны на применении ПО: Интернет и спецэффекты в кинематографе, медицинские системы жизнеобеспечения и космические программы, высокопроизводительный анализ финансовых данных и научные исследования. Эти, а также более традиционные проекты имеют много общего, поэтому применение улучшенных методов программирования окупится во всех случаях.

Признавая важность улучшения разработки ПО в целом, вы можете спросить: «Почему именно конструированию в этой книге уделяется такое внимание?»

Ответы на этот вопрос приведены ниже.

Конструирование — крупная часть процесса разработки ПО В зависимости от размера проекта на конструирование обычно уходит 30–80 % общего времени работы. Все, что занимает так много времени работы над проектом, неизбежно влияет на его успешность.

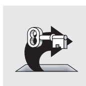
Перекрестная ссылка О связи между размером проекта и долей времени, уходящего на конструирование, см. подраздел «Соотношение между выполняемыми операциями и размер» раздела 27.5.

Конструирование занимает центральное место в процессе разработки ПО Требования к приложению и его архитектура разрабатываются до этапа конструирования, чтобы гарантировать его эффективность. Тестирование системы (в строгом смысле независимого тестирования) выполняется после конструирования и служит для проверки его правильности. Конструирование — центр процесса разработки ПО.

Перекрестная ссылка О производительности труда программистов см. подраздел «Индивидуальные различия» раздела 28.5.

Повышенное внимание к конструированию может намного повысить производительность труда отдельных программистов В своем классическом исследовании Сэкман, Эриксон и Грант показали, что производительность труда отдельных программистов во время конструирования изменяется в 10–20 раз (Sackman, Erikson, and Grant, 1968). С тех пор эти данные были подтверждены другими исследованиями (Curtis, 1981; Mills, 1983; Curtis et al., 1986; Card, 1987; Valett and McGarry, 1989; DeMarco and Lister, 1999№; Boehm et al., 2000). Эта книга поможет всем программистам изучить методы, которые уже используются лучшими разработчиками.

Результат конструирования — исходный код — часто является единственным верным описанием программы Зачастую единственным видом доступной программистам документации является сам исходный код. Спецификации требований и проектная документация могут устареть, но исходный код актуален всегда, поэтому он должен быть максимально качественным. Последовательное применение методов улучшения исходного кода — вот что отличает детальные, корректные и поэтому информативные программы от устройств Руба Голдберга[1]. Эффективнее всего применять эти методы на этапе конструирования.



Конструирование — единственный процесс, который выполняется во всех случаях Идеальный программный проект до начала конструирования проходит стадии тщательной выработки требований и проектирования архитектуры. После конструирования в идеале должно быть выполнено исчерпывающее, статистически контролируемое тестирование системы. Однако в реальных проектах нашего несовершенного мира разработчики часто пропускают этапы выработки требований и проектирования, начиная прямо с конструирования программы. Тестирование также часто выпадает из расписания из-за огромного числа ошибок и недостатка времени. Но каким бы срочным или плохо спланированным ни был проект, куда без конструирования деться? Так что повышение эффективности конструирования ПО позволяет оптимизировать любой проект, каким бы несовершенным он ни был.

1.3. Как читать эту книгу

Вы можете читать эту книгу от корки до корки или по отдельным темам. Если вы предпочитаете первый вариант, переходите к главе 2. Если второй — можете начать с главы 6 и переходить по перекрестным ссылкам к другим темам, которые вас заинтересуют. Если вы не уверены, какой из этих вариантов вам подходит, начните с раздела 3.2.

[1] Голдберг, Рубен Лушес («Руб») [Goldberg, «Rube» (Reuben Lucius)] (1883–1970) — карикатурист, скульптор. Известен своими карикатурами, в которых выдуманное им сложное оборудование («inventions») выполняет примитивные и никому не нужные операции. Лауреат Пулитцеровской премии 1948 г. за политические карикатуры. — *Прим. перев.*

Ключевые моменты

- Конструирование — главный этап разработки ПО, без которого не обходится ни один проект.
- Основные этапы конструирования: детальное проектирование, кодирование, отладка, интеграция и тестирование приложения разработчиками (блочное тестирование и интеграционное тестирование).
- Конструирование часто называют «кодированием» и «программированием».
- От качества конструирования во многом зависит качество ПО.
- В конечном счете ваша компетентность в конструировании ПО определяет то, насколько хороший вы программист. Совершенствованию ваших навыков и посвящена оставшаяся часть этой книги.

ГЛАВА 2

Метафоры, позволяющие лучше понять разработку ПО

http://cc2e.com/0278

Содержание

- 2.1. Важность метафор
- 2.2. Как использовать метафоры?
- 2.3. Популярные метафоры, характеризующие разработку ПО

Связанная тема

- Эвристика при проектировании: подраздел «Проектирование — эвристический процесс» в разделе 5.1

Терминология компьютерных наук — одна из самых красочных. Действительно, в какой еще области существуют стерильные комнаты с тщательно контролируемой температурой, заполненные вирусами, троянскими конями, червями, жучками и прочей живностью и нечистью?

Все эти яркие метафоры описывают специфические аспекты мира программирования. Более общие явления характеризуются столь же красочными метафорами, позволяющих лучше понять процесс разработки ПО.

Остальная часть книги не зависит от обсуждения метафор в этой главе. Можете пропустить ее, если хотите быстрее добраться до практических советов. Если хотите яснее представлять разработку ПО, читайте дальше.

2.1. Важность метафор

Проведение аналогий часто приводит к важным открытиям. Сравнив не совсем понятное явление с чем-то похожим, но более понятным, вы можете догадаться, как справиться с проблемой. Такое использование метафор называется «моделированием».

История науки полна открытий, сделанных благодаря метафорам. Так, химик Кекуле однажды во сне увидел змею, схватившую себя за хвост. Проснувшись, он понял, что свойства бензола объяснила бы молекулярная структура, имеющая похожую кольцевую форму. Дальнейшие эксперименты подтвердили его гипотезу (Barbour, 1966).

Кинетическая теория газов была создана на основе модели «бильярдных шаров», согласно которой молекулы газа, подобно бильярдным шарам, имеют массу и совершают упругие соударения.

Волновая теория света была разработана преимущественно путем исследования сходств между светом и звуком. И свет, и звук имеют амплитуду (яркость — громкость), частоту (цвет — высота) и другие общие свойства. Сравнение волновых теорий звука и света оказалось столь продуктивным, что ученые потратили много сил, пытаясь обнаружить среду, которая распространяла бы свет, как воздух распространяет звук. Они даже дали этой среде название — «эфир», но так и не смогли ее обнаружить. В данном случае аналогия была такой убедительной, что ввела ученых в заблуждение.

В целом эффективность моделей объясняется их яркостью и концептуальной целостностью. Модели подсказывают ученым свойства, отношения и перспективные области исследований. Иногда модели вводят в заблуждение; как правило, к этому приводит чрезмерное обобщение метафоры. Поиск эфира — наглядный пример чрезмерного обобщения модели.

Разумеется, некоторые метафоры лучше других. Хорошими метафорами можно считать те, что отличаются простотой, согласуются с другими релевантными метафорами и объясняют многие экспериментальные данные и наблюдаемые явления.

Рассмотрим, к примеру, колебания камня, подвешенного на веревке. До Галилея сторонники Аристотеля считали, что тяжелый объект перемещается из верхней точки в нижнюю, переходя в состояние покоя. В данном случае они подумали бы, что камень падает, но с осложнениями. Когда Галилей смотрел на раскачивающийся камень, он видел маятник, поскольку камень снова и снова повторял одно и то же движение.

Указанные модели фокусируют внимание на совершенно разных факторах. Последователи Аристотеля, рассматривавшие раскачивающийся камень как падающий объект, принимали в расчет вес камня, высоту, на которую он был поднят, и время, проходящее до достижения камнем состояния покоя. В модели Галилея важными были другие факторы. Он обращал внимание на вес камня, угловое смещение, радиус и период колебаний маятника. Благодаря этому Галилей открыл законы, которые последователи Аристотеля открыть не смогли, так как их модель заставила их наблюдать за другими явлениями и задавать другие вопросы.

Аналогичным образом метафоры способствуют и лучшему пониманию вопросов разработки ПО. Во время лекции по случаю получения премии Тьюринга в 1973 г., Чарльз Бахман (Charles Bachman) упомянул переход от доминировавшего геоцентрического представления о Вселенной к гелиоцентрическому. Геоцентрическая модель Птолемея не вызывала почти никаких сомнений целых 1400 лет. Затем, в 1543 г., Коперник выдвинул гелиоцентрическую теорию, предположив, что центром Вселенной на самом деле является Солнце, а не Земля. В конечном итоге такое изменение умозрительных моделей привело к открытию новых планет, исключению Луны из категории планет и переосмыслению места человечества во Вселенной.

> Не стоит недооценивать важность метафор. Метафоры имеют одно неоспоримое достоинство: описываемое ими поведение предсказуемо и понятно всем людям. Это сокращает объем ненужной коммуникации, способствует достижению взаимопонимания и ускоряет обучение. По сути метафора — это способ осмысления и абстрагирования концепций, позволяющий думать в более высокой плоскости и избегать низкоуровневых ошибок.
>
> *Фернандо Дж. Корбати (Fernando J. Corbaty)*

Переход от геоцентрического представления к гелиоцентрическому в астрономии Бахман сравнил с изменением, происходившим в программировании в начале 1970-х. В это время центральное место в моделях обработки данных стали отводить не компьютерам, а базам данных. Бахман указал, что создатели ранних моделей стремились рассматривать все данные как последовательный поток карт, «протекающий» через компьютер (компьютеро-ориентированный подход). Суть изменения заключалась в отведении центрального места пулу данных, над которыми компьютер выполняет некоторые действия (подход, ориентированный на БД).

Сегодня почти невозможно найти человека, считающего, что Солнце вращается вокруг Земли. Столь же трудно представить программиста, который бы думал, что все данные можно рассматривать как последовательный поток карт. После опровержения старых теорий трудно понять, как кто-то когда-то мог в них верить. Еще удивительнее то, что приверженцам этих старых теорий новые теории казались такими же нелепыми, как нам старые.

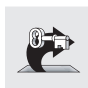
Геоцентрическое представление о Вселенной мешало астрономам, которые сохранили верность этой теории после появления лучшей. Аналогично компьютеро-ориентированное представление о компьютерной вселенной тянуло назад компьютерных ученых, которые продолжили придерживаться его после появления теории, ориентированной на БД.

Иногда люди упрощают суть метафор. На каждый из описанных примеров так и тянет ответить: «Разумеется, правильная метафора более полезна. Другая метафора была неверной!» Так-то оно так, но это слишком упрощенное представление. История науки — не серия переходов от «неверных» метафор к «верным». Это серия переходов от «менее хороших» метафор к «лучшим», от менее полных теорий к более полным, от адекватного описания одной области к адекватному описанию другой.

В действительности многие модели, на смену которым приходят лучшие модели, сохраняют свою полезность. Так, инженеры до сих пор решают большинство своих задач, опираясь на ньютонову динамику, хотя в теоретической физике ее вытеснила теория Эйнштейна.

Разработка ПО — относительно молодая область науки. Она еще недостаточно зрелая, чтобы иметь набор стандартных метафор. Поэтому она включает массу второстепенных и противоречивых метафор. Одни из них лучше другие — хуже. Оттого, насколько хорошо вы понимаете метафоры, зависит и ваше понимание разработки ПО.

2.2. Как использовать метафоры?



Метафора, характеризующая разработку ПО, больше похожа на прожектор, чем на дорожную карту. Она не говорит, где найти ответ — она говорит, как его искать. Метафора — это скорее эвристический подход, а не алгоритм.

Алгоритмом называют последовательность четко определенных команд, которые необходимо выполнить для решения конкретной задачи. Алгоритм предсказуем, детерминирован и не допускает случайностей. Алгоритм говорит, как пройти из точки А в точку Б не дав крюку, без посещения точек В, Г и Д и без остановок на чашечку кофе.

Эвристика — это метод, помогающий искать ответ. Результаты его применения могут быть в некоторой степени случайными, потому что эвристика указывает только способ поиска, но не говорит, что искать. Она не говорит, как дойти прямо из точки А в точку Б; даже положение этих точек может быть неизвестно. Эвристика — это алгоритм в шутовском наряде. Она менее предсказуема, более забавна и поставляется без 30-дневной гарантии с возможностью возврата денег.

Вот алгоритм, позволяющий добраться до чьего-то дома: поезжайте по шоссе 167 на юг до городка Пюиолап. Сверните на аллею Сауз-Хилл, а дальше 4,5 мили вверх по холму. Поверните у продуктового магазина направо, а на следующем перекрестке — налево. Доехав до дома 714, расположенного на левой стороне улицы, остановитесь и выходите из автомобиля.

А эвристическое правило может быть таким: найдите наше последнее письмо. Езжайте в город, указанный на конверте. Оказавшись в этом городе, спросите кого-нибудь, где находится наш дом. Все нас знают — кто-нибудь с радостью вам поможет. Если никого не встретите, позвоните нам из телефона-автомата, и мы за вами приедем.

Перекрестная ссылка Об использовании эвристики при проектировании ПО см. подраздел «Проектирование — эвристический процесс» раздела 5.1.

Различия между алгоритмом и эвристикой тонки, и в чем-то эти два понятия перекрываются. В этой книге основным различием между ними я буду считать степень косвенности решения. Алгоритм предоставляет вам сами команды. Эвристика сообщает вам, как обнаружить команды самостоятельно или по крайней мере где их искать.

Наличие директив, точно определяющих способ решения проблем программирования, непременно сделало бы программирование более легким, а результаты более предсказуемыми. Но наука программирования еще не продвинулась так далеко, а может, этого никогда и не случится. Самая сложная часть программирования — концептуализация проблемы, и многие ошибки программирования являются концептуальными. С концептуальной точки зрения каждая программа уникальна, поэтому трудно или даже невозможно разработать общий набор директив, приводящих к решению во всех случаях. Так что знание общего подхода к проблемам не менее, а то и более ценно, чем знание точных решений конкретных проблем.

Как использовать метафоры, характеризующие разработку ПО? Используйте так, чтобы лучше разобраться в проблемах и процессах программирования, чтобы они помогали вам размышлять о выполняемых действиях и придумывать лучшие решения задач. Конечно, вы не сможете взглянуть на строку кода и сказать, что она противоречит одной из метафор, описываемых в этой главе. Но со временем тот, кто использует метафоры, лучше поймет программирование и будет быстрее создавать более эффективный код, чем тот, кто их не использует.

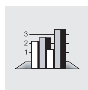
2.3. Популярные метафоры, характеризующие разработку ПО

Множество метафор, описывающих разработку ПО, смутит кого угодно. Дэвид Грайс утверждает, что написание ПО — это наука (Gries, 1981). Дональд Кнут считает это искусством (Knuth, 1998). Уоттс Хамфри говорит, что это процесс (Humphrey, 1989). Ф. Дж. Плоджер и Кент Бек утверждают, что разработка ПО похожа на управление автомобилем, однако приходят к почти противоположным выводам (Plauger, 1993, Beck, 2000). Алистер Кокберн сравнивает разработку ПО с игрой (Cockburn, 2002), Эрик Реймонд — с базаром (Raymond, 2000), Энди Хант (Andy Hunt) и Дэйв Томас (Dave Thomas) — с работой садовника, Пол Хекель — со съемкой фильма «Белоснежка и семь гномов» (Heckel, 1994). Фред Брукс упоминает фермерство, охоту на оборотней и динозавров, завязших в смоляной яме (Brooks, 1995). Какие метафоры самые лучшие?

Литературная метафора: написание кода

Самая примитивная метафора, описывающая разработку ПО, берет начало в выражении «написание кода». Согласно литературной метафоре разработка программы похожа на написание письма: вы садитесь за стол, берете бумагу, перо и пишете письмо с начала до конца. Это не требует никакого формального планирования, а мысли, выражаемые в письме, формулируются автором по ходу дела.

На этой метафоре основаны и многие другие идеи. Джон Бентли (Jon Bentley) говорит, что программист должен быть способен сесть у камина со стаканом бренди, хорошей сигарой, охотничьей собакой у ног и «сформулировать программу» подобно тому, как писатели создают романы. Брайан Керниган и Ф. Дж. Плоджер назвали свою книгу о стиле программирования «The Elements of Programming Style» (Kernighan and Plauger, 1978), обыгрывая название книги о литературном стиле «The Elements of Style» (Strunk and White, 2000). Программисты часто говорят об «удобочитаемости программы».



Индивидуальную работу над небольшими проектами метафора написания письма характеризует довольно точно, но в целом она описывает разработку ПО неполно и неадекватно. Письма и романы обычно принадлежат перу одного человека, тогда как над программами обычно работают группы людей с разными сферами ответственности. Закончив писать письмо, вы запечатываете его в конверт и отправляете. С этого момента изменить вы его не можете, и письмо во всех отношениях является завершенным. Изменить ПО не так уж трудно, и вряд ли работу над ним можно когда-нибудь признать законченной. Из общего объема работы над типичной программной системой две трети обычно выполняются после выпуска первой версии программы, а иногда эта цифра достигает целых 90 % (Pigoski, 1997). В литературе поощряется оригинальность. При конструировании ПО оригинальный подход часто оказывается менее эффективным, чем повторное использование идей, кода и тестов из предыдущих проектов. Словом, процесс разработки ПО, соответствующий литературной метафоре, является слишком простым и жестким, чтобы быть полезным.

К сожалению, литературная метафора была увековечена в одной из самых популярных книг по разработке ПО — книге Фреда Брукса «The Mythical Man-Month» («Мифический человеко-месяц») (Brooks, 1995). Брукс пишет: «Планируйте выбросить первый экземпляр программы: вам в любом случае придется это сделать». Перед глазами невольно возникает образ мусорного ведра, полного черновиков (рис. 2-1).

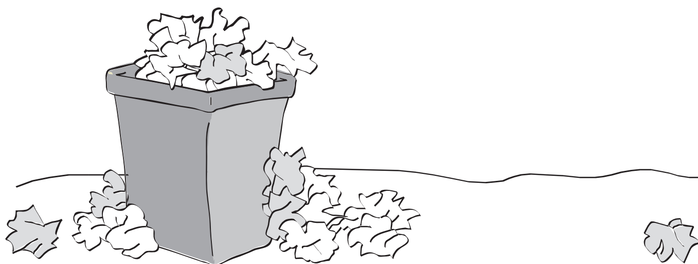


Рис. 2-1. *Литературная метафора наводит на мысль, что процесс разработки ПО основан на дорогостоящем методе проб и ошибок, а не на тщательном планировании и проектировании*

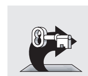
Подобный подход может быть практичным, если вы пишете банальное письмо своей тетушке. Однако расширение метафоры «написания» ПО вплоть до выбрасывания первого экземпляра программы — не лучший совет в мире разработки ПО, где крупная система по стоимости уже сравнялась с 10-этажным офисным зданием или океанским лайнером. Конечно, это не имело бы значения, если б вы имели бесконечные запасы времени и средств. Однако реальные условия таковы, что разработчики должны создавать программы с первого раза или хотя бы минимизировать объем дополнительных расходов в случае неудач. Другие метафоры лучше иллюстрируют достижение таких целей.

> Планируйте выбросить первый экземпляр программы: вам в любом случае придется это сделать.
>
> *Фред Брукс*

> Если вы планируете выбросить первый экземпляр программы, вы выбросите и второй.
>
> *Крейг Зеруни (Craig Zerouni)*

Сельскохозяйственная метафора: выращивание системы

Некоторые разработчики заявляют, что создание ПО следует рассматривать по аналогии с выращиванием сельскохозяйственных культур. Вы проектируете отдельный блок, кодируете его, тестируете и добавляете в систему, чуть расширяя с каждым разом ее функциональность. Такое разбиение задачи на множество небольших действий позволяет минимизировать проблемы, с которыми можно столкнуться на каждом этапе.



Иногда хороший метод описывается плохой метафорой. В таких случаях попытайтесь сохранить метод и обнаружить лучшую метафору. В данном случае инкрементный подход полезен, но сельскохозяйственная метафора просто ужасна.

Дополнительные сведения О другой сельскохозяйственной метафоре, употребляемой в контексте сопровождения ПО, см. главу «On the Origins of Designer Intuition» книги «Rethinking Systems Analysis and Design» (Weinberg, 1988).

Возможно, идея выполнения небольшого объема работы зараз и напоминает рост растений, но сельскохозяйственная аналогия неубедительна и неинформативна, к тому же ее легко заменить лучшими метафорами, которые описаны ниже. Сельскохозяйственную метафору трудно расширить за пределы идеи выполнения чего-либо небольшими частями. Ее приверженцы (рис. 2-2) рискуют в итоге заговорить об удобрении плана системы, прореживании детального проекта, повышении урожайности кода путем оптимизации землеустройства и уборке урожая самого кода. Вы начнете говорить о чередовании посевов C++ и о том, что было бы неплохо оставить систему под паром для повышения концентрации азота на жестком диске.

Слабость данной метафоры заключается в предположении, что у вас нет прямого контроля над развитием ПО. Вы просто сеете семена кода весной и, если на то будет воля Великой Тыквы, осенью получите невиданный урожай кода.

Рис. 2-2. *Нелегко адекватно расширить сельскохозяйственную метафору на область разработки ПО*

Метафора жемчужины: медленное приращение системы

Иногда, говоря о выращивании ПО, на самом деле имеют в виду приращение, или аккрецию (accretion). Две этих метафоры тесно связаны, но вторая более убедительна. Приращение характеризует процесс формирования жемчужины за счет отложения небольших объемов карбоната кальция. В геологии и юриспруденции под аккрецией понимают увеличение территории суши посредством отложения содержащихся в воде пород.

Перекрестная ссылка О применении инкрементных стратегий при интеграции системы см. раздел 29.2.

Это не значит, что вы должны освоить создание кода из осадочных пород; это означает, что вы должны научиться добавлять в программные системы по небольшому фрагменту за раз. Другими словами, которые в связи с этим приходят на ум, являются термины «инкрементный», «итеративный», «адаптивный» и «эволюционный». Инкрементное проектирование, конструирование и тестирование — одни из самых эффективных концепций разработки ПО.

При инкрементной разработке вы сначала создаете самую простую версию системы, которую можно было бы запустить. Она может не принимать реальных данных, может не выполнять над ними реальных действий, может не генерировать реальные результаты — она должна быть просто скелетом, достаточно крепким,

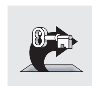
чтобы поддерживать реальную систему по мере ее разработки. Она может вызывать поддельные классы для каждой из определенных вами основных функций. Такая система похожа на песчинку, с которой начинается образование жемчужины.

Создав скелет, вы начинаете понемногу наращивать плоть. Каждый из фиктивных классов вы заменяете реальным. Вместо того чтобы имитировать ввод данных, вы пишете код, на самом деле принимающий реальные данные. А вместо имитации вывода данных — код, на самом деле выводящий данные. Вы продолжаете добавлять нужные фрагменты, пока не получаете полностью рабочую систему.

Эффективность такого подхода можно подтвердить двумя впечатляющими примерами. Фред Брукс, который в 1975 г. предлагал выбрасывать первый экземпляр программы, заявил, что за десять лет, прошедших с момента написания им знаменитой книги «Мифический человеко-месяц», ничто не изменяло его работу и ее эффективность так радикально, как инкрементная разработка (Brooks, 1995). Аналогичное заявление было сделано Томом Гилбом в революционной книге «Principles of Software Engineering Management» (Gilb, 1988), в которой он представил метод эволюционной поставки программы (evolutionary delivery) и разработал многие основы современного гибкого программирования (agile programming). Многие другие современные методологии также основаны на идее инкрементной разработки (Beck, 2000; Cockburn, 2002; Highsmith, 2002; Reifer, 2002; Martin, 2003; Larman, 2004).

Достоинство инкрементной метафоры в том, что она не дает чрезмерных обещаний. Кроме того, она не так легко поддается неуместному расширению, как сельскохозяйственная метафора. Раковина, формирующая жемчужину, — хороший вариант визуализации инкрементной разработки, или аккреции.

Строительная метафора: построение ПО



Метафора «построения» ПО полезнее, чем метафоры «написания» или «выращивания» ПО, так как согласуется с идеей аккреции ПО и предоставляет более детальное руководство. Построение ПО подразумевает наличие стадий планирования, подготовки и выполнения, тип и степень выраженности которых зависят от конкретного проекта. При изучении этой метафоры вы найдете и другие параллели.

Для построения метровой башни требуется твердая рука, ровная поверхность и 10 пивных банок, для башни же в 100 раз более высокой недостаточно иметь в 100 раз больше пивных банок. Такой проект требует совершенно иного планирования и конструирования.

Если вы строите простой объект, скажем, собачью конуру, вы можете пойти в хозяйственный магазин, купить доски, гвозди, и к вечеру у Фидо будет новый дом. Если вы забудете про лаз или допустите какую-нибудь другую ошибку, ничего страшного: вы можете ее исправить или даже начать все сначала (рис. 2-3). Все, что вы при этом потеряете, — время. Такой свободный подход уместен и в небольших программных проектах. Если вы плохо спроектируете 1000 строк кода, то сможете выполнить рефакторинг или даже начать проект заново, и это не приведет к крупным потерям.

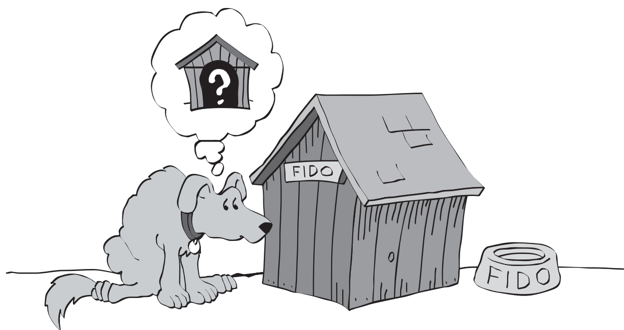


Рис. 2-3. *За ошибку, допущенную при создании простого объекта, приходится расплачиваться лишь потраченным временем и, возможно, некоторым разочарованием*

Построить дом сложнее, и плохое проектирование при этом приводит к куда более серьезным последствиям. Сначала вы должны решить, какой тип здания вы хотите построить, что аналогично определению проблемы при разработке ПО. Затем вы с архитектором должны разработать и утвердить общий план, что похоже на разработку архитектуры. Далее вы чертите подробные чертежи и нанимаете бригаду строителей — это аналогично детальному проектированию ПО. Вы готовите стройплощадку, закладываете фундамент, создаете каркас дома, обшиваете его, кроете крышу и проводите в дом все коммуникации — это похоже на конструирование ПО. Когда строительство почти завершено, в дело вступают ландшафтные дизайнеры, маляры и декораторы, делающие дом максимально удобным и привлекательным. Это напоминает оптимизацию ПО. Наконец, на протяжении всего строительства вас посещают инспекторы, проверяющие стройплощадку, фундамент, электропроводку и все, что можно проверить. При разработке ПО этому соответствуют обзоры и инспекция проекта.

И в строительстве, и в программировании увеличение сложности и масштаба проекта сопровождается ростом цены ошибок. Конечно, для создания дома нужны довольно дорогие материалы, однако главной статьей расходов является оплата труда рабочих. Перемещение стены на 15 см обойдется дорого не потому, что при этом будет потрачено много гвоздей, а потому, что вам придется оплатить дополнительное время работы строителей. Чтобы не тратить время на исправление ошибок, которых можно избежать, вы должны как можно лучше выполнить проектирование (рис. 2-4). Материалы, необходимые для создания программного продукта, стоят дешевле, чем стройматериалы, однако затраты на рабочую силу в обоих случаях примерно одинаковы. Изменение формата отчета обходится ничуть не дешевле, чем перемещение стены дома, потому что главным компонентом затрат в обоих случаях является время людей.

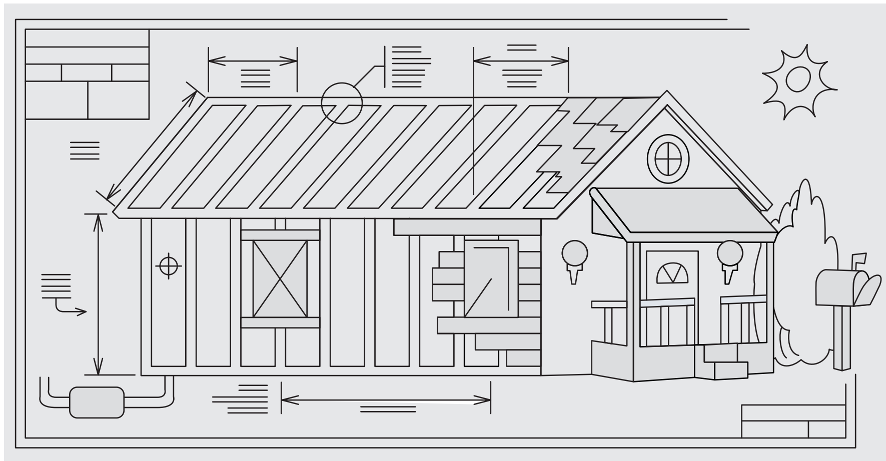


Рис. 2-4. *Более сложные объекты требуют более тщательного планирования*

Какие еще параллели можно провести между сооружением дома и разработкой ПО? При возведении дома никто не пытается конструировать вещи, которые можно купить. Здравомыслящему человеку и в голову не придет самостоятельно разрабатывать и создавать стиральную машину, холодильник, шкафы, окна и двери, если все это можно приобрести. Создавая программную систему, вы поступите так же. Вы будете в полной мере использовать возможности высокоуровневого языка вместо того, чтобы писать собственный код на уровне ОС. Возможно, вы используете также встроенные библиотеки классов-контейнеров, научные функции, классы пользовательского интерфейса и классы для работы с БД. Обычно невыгодно писать компоненты, которые можно купить готовыми.

Однако если вы хотите построить нестандартный дом с первоклассной меблировкой, мебель, возможно, придется заказать. Вы можете заказать встроенные посудомоечную машину и холодильник, чтобы они выглядели как часть обстановки. Вы можете заказать окна необычных форм и размеров. Такое изготовление предметов на заказ имеет параллели и в мире разработки ПО. При работе над приложением высшего класса для достижения более высокой скорости и точности расчетов или реализации необычного интерфейса иногда приходится создавать собственные научные функции, собственные классы-контейнеры и другие компоненты.

И конструирование дома, и конструирование ПО можно оптимизировать, выполнив адекватное планирование. Если создавать ПО в неверном порядке, его будет трудно писать, тестировать и отлаживать. Сроки затянутся, да и весь проект может завершиться неудачей из-за чрезмерной сложности отдельных компонентов, не позволяющей разработчикам понять работу всей системы.

Тщательное планирование не значит исчерпывающее или чрезмерное. Вы можете спланировать основные структурные компоненты и позднее решать, чем покрыть пол, в какой цвет окрасить стены, какой использовать кровельный материал и т. д. Хорошо спланированный проект открывает больше возможностей для изменения решения на более поздних этапах работы. Чем лучше вам известен тип создаваемого ПО, тем больше деталей вы можете принимать как данное. Вы про-

сто должны убедиться в проведении достаточного планирования, чтобы его недостаток не привел позднее к серьезным проблемам.

Аналогия конструирования также помогает понять, почему разные программные проекты призывают к разным подходам разработки. Склад и медицинский центр или ядерный реактор также требуют разных степеней планирования, проектирования и контроля качества, и никто не стал бы одинаково подходить к строительству школы, небоскреба и жилого дома с тремя спальнями. Работая над ПО, вы обычно можете использовать гибкие упрощенные подходы, но иногда для обеспечения безопасности и других целей необходимы и жесткие, тщательно продуманные подходы.

Проблема изменения ПО приводит нас к еще одной параллели. Перемещение несущей стены на 15 см обходится гораздо дороже, чем перемещение перегородки между комнатами. Аналогично внесение структурных изменений в программу требует больших затрат, чем добавление или удаление второстепенных возможностей.

Наконец, проведение аналогии с домостроительством позволяет лучше понять работу над очень крупными программными проектами. При создании очень крупного объекта цена неудачи слишком высока, поэтому объект надо спроектировать тщательнейшим образом. Строительные организации скрупулезно разрабатывают и инспектируют свои планы. Все крупные здания создаются с большим запасом прочности; лучше заплатить на 10 % больше за более прочный материал, чем рисковать крушением небоскреба. Кроме того, большое внимание уделяется времени. При возведении Эмпайр Стейт Билдинг время прибытия каждого грузовика, поставлявшего материалы, задавалось с точностью до 15 минут. Если грузовик не прибывал в нужное время, задерживалась работа над всем проектом.

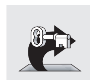
Аналогично, очень крупные программные проекты требуют планирования более высокого порядка, чем просто крупные проекты. Кейперс Джонс сообщает, что программная система из одного миллиона строк кода требует в среднем 69 *видов* документации (Jones, 1998). Спецификация требований к такой системе обычно занимает 4000–5000 страниц, а проектная документация вполне может быть еще в 2 или 3 раза более объемной. Маловероятно, чтобы один человек мог понять весь проект такого масштаба или даже прочитать всю документацию, поэтому подобные проекты требуют более тщательной подготовки.

По экономическому масштабу некоторые программные проекты сравнимы с возведением «Эмпайр Стейт Билдинг», и контролироваться они должны соответствующим образом.

Дополнительные сведения Грамотные комментарии по поводу расширения метафоры конструирования см. в статье «What Supports the Roof?» (Starr 2003).

Метафора построения-конструирования может быть расширена во многих других направлениях, именно поэтому она столь эффективна. Благодаря этой метафоре отрасль разработки ПО обогатилась многими популярными терминами, такими как архитектура ПО, леса (scaffolding), конструирование и фундаментальные классы. Наверное, вы сможете назвать и другие примеры.

Применение методов разработки ПО: интеллектуальный инструментарий

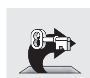


Люди, эффективно разрабатывающие высококачественное ПО, многие годы посвятили сбору методов, хитростей и магических заклинаний. Эти методы — не правила, а аналитические инструменты. Хороший рабочий знает предназначение каждого инструмента и умеет эффективно его использовать. Программисты не исключение. Чем больше вы узнаете о программировании, тем больше аналитических инструментов и знаний об их своевременном и правильном использовании накапливается в вашем интеллектуальном инструментарии.

Консультанты по вопросам разработки ПО иногда советуют программистам придерживаться одних методов разработки ПО в ущерб другим. Это печально, потому что, если вы станете использовать только одну методологию, вы увидите весь мир в терминах этой методологии. В некоторых случаях это сделает недоступными для вас другие методы, лучше подходящие для решения текущей проблемы. Метафора инструментария поможет вам держать все методы, способы и хитрости в пределах досягаемости и применять их в уместных обстоятельствах.

Перекрестная ссылка О выборе методов проектирования и их комбинировании см. раздел 5.3.

Комбинирование метафор



Метафоры имеют эвристическую, а не алгоритмическую природу, поэтому они не исключают друг друга. Вы можете использовать и метафору аккреции, и метафору конструирования. Если хотите, можете представлять разработку ПО как написание письма, комбинируя эту метафору с вождением автомобиля, охотой на оборотней или образом динозавра, увязшего в смоляной луже. Используйте любые метафоры или их комбинации, которые стимулируют ваше мышление или помогают общаться с другими членами группы.

Использование метафор — дело тонкое. Чтобы метафора привела вас к ценным эвристическим догадкам, вы должны ее расширить. Но если ее расширить чересчур или в неверном направлении, она может ввести в заблуждение. Как и любой мощный инструмент, метафоры можно использовать неверным образом, однако благодаря своей мощи они могут стать ценным компонентом вашего интеллектуального инструментария.

Дополнительные ресурсы

http://cc2e.com/0285

Среди книг общего плана, посвященных метафорам, моделям и парадигмам, главное место занимает «The Structure of Scientific Revolutions» (3d ed. Chicago, IL: The University of Chicago Press, 1996) Томаса Куна (Thomas S. Kuhn). В своей книге, увидевшей свет в 1962 г., Кун рассказывает о возникновении, развитии и смене теорий. Этот труд, вызвавший множество споров по вопросам философии науки, отличается ясностью, лаконичностью и включает массу интересных примеров взлетов и падений научных метафор, моделей и парадигм.

Статья «The Paradigms of Programming». 1978 Turing Award Lecture («Communications of the ACM», August 1979, pp. 455–60) Роберта У. Флойда (Robert W. Floyd) представляет собой увлекательное обсуждение использования моделей при разработке ПО; некоторые аспекты рассматриваются в ней в контексте идей Томаса Куна.

Ключевые моменты

- Метафоры являются по природе эвристическими, а не алгоритмическими, поэтому зачастую они немного небрежны.
- Метафоры помогают понять процесс разработки ПО, сопоставляя его с другими, более знакомыми процессами.
- Некоторые метафоры лучше, чем другие.
- Сравнение конструирования ПО с возведением здания указывает на необходимость тщательной подготовки к проекту и проясняет различие между крупными и небольшими проектами.
- Аналогия между методами разработки ПО и инструментами в интеллектуальном инструментарии программиста наводит на мысль, что в распоряжении программистов имеется множество разных инструментов и что ни один инструмент не является универсальным. Выбор правильного инструмента — одно из условий эффективного программирования.
- Метафоры не исключают друг друга. Используйте комбинацию метафор, наиболее эффективную в вашем случае.

ГЛАВА 3

Семь раз отмерь, один раз отрежь: предварительные условия

http://cc2e.com/0309

Содержание

- 3.1. Важность выполнения предварительных условий
- 3.2. Определите тип ПО, над которым работаете
- 3.3. Предварительные условия, связанные с определением проблемы
- 3.4. Предварительные условия, связанные с выработкой требований
- 3.5. Предварительные условия, связанные с разработкой архитектуры
- 3.6. Сколько времени посвятить выполнению предварительных условий?

Связанные темы

- Основные решения, которые приходится принимать при конструировании: глава 4
- Влияние размера проекта на предварительные условия и процесс конструирования ПО: глава 27
- Связь между качеством ПО и аспектами его конструирования: глава 20
- Управление конструированием ПО: глава 28
- Проектирование ПО: глава 5

Перед началом конструирования дома строители просматривают чертежи, проверяют, все ли разрешения получены, и исследуют фундамент. К сооружению небоскреба, жилого дома и собачьей конуры строители готовились бы по-разному, но, каким бы ни был проект, перед началом конструирования они провели бы добросовестную подготовку с учетом всех особенностей проекта.

В этой главе мы рассмотрим компоненты подготовки к конструированию ПО. Как и в строительстве, конечный успех программного проекта во многом определяется до начала конструирования. Если фундамент ненадежен или планирование выполнено небрежно, на этапе конструирования вы в лучшем случае сможете только свести вред к минимуму.

Популярная у плотников поговорка «семь раз отмерь, один раз отрежь» очень актуальна на этапе конструирования ПО, затраты на который иногда составляют аж 65% от общего бюджета проекта. В неудачных программных проектах конструирование иногда приходится выполнять дважды, трижды и даже больше. Как и в любой другой отрасли, повторение самой дорогостоящей части программного проекта ни к чему хорошему привести не может.

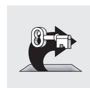
Хотя чтение этой главы является залогом успешного конструирования ПО, само конструирование в ней не обсуждается. Если вы уже хорошо разбираетесь в цикле разработки ПО или вам не терпится добраться до обсуждения конструирования, можете перейти к главе 5. Если вам не нравится идея выполнения предварительных условий конструирования, просмотрите раздел 3.2, чтобы узнать, какую роль они играют в вашем случае, а затем вернитесь к разделу 3.1, в котором описываются расходы, связанные с их невыполнением.

3.1. Важность выполнения предварительных условий

Перекрестная ссылка Повышение внимания к качеству ПО — самый эффективный способ повышения производительности труда программистов. (см. раздел 20.5).

Общей чертой всех программистов, создающих высококачественное ПО, является использование высококачественных методов, ставящих ударение на качестве ПО в самом начале, середине и конце проекта.

Если вы подчеркиваете качество в конце проекта, это приходится на этап тестирования системы. Именно о тестировании думают многие люди, представляя процесс гарантии качества ПО, но тестирование — только один из компонентов стратегии гарантии качества, и не самый важный. Тестирование не позволяет обнаружить такие ошибки, как создание не того приложения или создание нужного приложения не тем образом; эти ошибки должны быть определены и устранены раньше — до начала конструирования.



Если вы уделяете повышенное внимание качеству в середине работы над проектом, вы подчеркиваете методы конструирования, которым и посвящена большая часть этой книги.

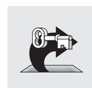
Если вы подчеркиваете качество в начале проекта, вы качественно выполняете планирование, определение требований и проектирование. Если вы спроектировали автомобиль «Понтиак Ацтек», то сколько бы вы его ни тестировали, он никогда не превратится в «Роллс-Ройс». Вы можете создать самый лучший «Ацтек», но если вам нужен «Роллс-Ройс», это нужно планировать с самого начала. При разработке ПО такому планированию соответствуют определение проблемы и определение и проектирование решения.

Конструирование — средний этап работы, поэтому ко времени начала конструирования успех проекта уже частично предопределен. И все же во время конструирования вы хотя бы должны быть в состоянии определить, насколько благополучна ваша ситуация, и вернуться назад, если на горизонте показались черные тучи неудачи. В оставшейся части этой главы я подробно расскажу, почему адекватная подготовка к конструированию так важна и как определить, действительно ли вы готовы перейти к нему.

Актуальны ли предварительные условия для современных программных проектов?

> Выбор методологии не должен быть невежественным. Она должна быть основана на самом новом и эффективном и дополнена старым и заслуживающим доверия.
>
> *Харлан Миллз (Harlan Mills)*

Порой говорят, что предварительные действия, такие как разработка архитектуры, проектирование и планирование проекта, в современных условиях бесполезны. Такие заявления не подтверждаются ни прошлыми, ни современными исследованиями (подробности см. ниже). Оппоненты предварительных условий обычно приводят примеры неудачного выполнения предварительных условий и делают вывод, что такая работа неэффективна. Тем не менее подготовку к конструированию можно выполнить успешно, и данные, накопленные с 1970-х, свидетельствуют о том, что в таких случаях работа над проектом оказывается эффективнее.



Общая цель подготовки — снижение риска: адекватное планирование позволяет исключить главные аспекты риска на самых ранних стадиях работы, чтобы основную часть проекта можно было выполнить максимально эффективно. Безусловно, главные факторы риска в создании ПО — неудачная выработка требований и плохое планирование проекта, поэтому подготовка направлена в первую очередь на оптимизацию этих этапов.

Так как подготовка к конструированию не является точной наукой, специфический подход к снижению риска будет в значительной степени определяться особенностями проекта (см. раздел 3.2).

Причины неполной подготовки

Возможно, вам кажется, что все профессионалы знают о важности подготовки и всегда до начала конструирования проверяют выполнение предварительных условий. Увы, это не так.

> **Дополнительные сведения** О профессиональной программе разработки ПО, поощряющей применение этих навыков, см. главу 16 книги «Professional Software Development» (McConnell, 2004).

> http://cc2e.com/0316

Зачастую причина неполной подготовки к конструированию ПО объясняется тем, что отвечающие за нее разработчики не имеют нужного опыта. Для планирования проекта, создания адекватной бизнес-модели, разработки полных и точных требований и высококачественной архитектуры нужно обладать далеко не тривиальными навыками, однако большинство разработчиков этому не обучены. Если разработчики не знают, как выполнять предварительную работу, рекомендация «выполнять больше такой работы» не имеет смысла: если работа изначально выполняется некачественно, ее выполнение в *больших* объемах не принесет никакой пользы! Объяснение выполнения этих действий не является предметом данной книги, однако в разделе «Дополнительные ресурсы» в конце главы я привел массу источников, позволяющих получить такой опыт.

Некоторые программисты умеют готовиться к конструированию, но пренебрегают подготовкой, потому что не могут устоять перед искушением пораньше приступить к кодированию. Если вы принадлежите к их числу, могу дать два совета. Первый: прочитайте следующий раздел. Возможно, у вас откроются глаза на не-

которые вещи. Второй: уделяйте внимание проблемам, с которыми сталкиваетесь. Поработав над несколькими крупными программами, вы прекрасно поймете пользу заблаговременного планирования. Положитесь на свой опыт.

Наконец, еще одна причина пренебрежения подготовкой к конструированию состоит в том, что менеджеры прохладно относятся к программистам, которые тратят на это время. Это довольно странно: такие люди, как Барри Бом (Barry Boehm), Гради Буч (Grady Booch) и Карл Вигерс (Karl Wiegers), отстаивают важность выработки требований и проектирования уже 25 лет, и менеджеры, казалось бы, уже должны понимать, что разработка ПО не ограничивается кодированием, но...

Дополнительные сведения Ряд интересных вариаций на эту тему см. в классическом труде Джеральда Вайнберга «The Psychology of Computer Programming» (Weinberg, 1998).

Несколько лет назад я работал над проектом Минобороны, и как-то на этапе выработки требований нас посетил куратор проекта — генерал. Мы сказали ему, что работаем над требованиями: большей частью общаемся с клиентами, определяем их потребности и разрабатываем проект приложения. Он, однако, настаивал на том, чтобы увидеть код. Мы сказали, что у нас нет кода, и тогда он отправился в рабочий отдел, намереваясь хоть кого-нибудь из 100 человек поймать за программированием. Огорченный тем, что почти все из них находились не за своими компьютерами, этот крупный человек наконец указал на инженера рядом со мной и проревел: «А он что делает? Он ведь пишет код!» Вообще-то этот инженер работал над утилитой форматирования документов, но генерал хотел увидеть код, нашел что-то похожее на него и хотел, чтобы хоть кто-то писал код, так что мы сказали ему, что он прав: это код.

Этот феномен известен как синдром WISCA или WIMP: «Why Isn't Sam Coding Anything? (Почему Сэм не пишет код?)» или «Why Isn't Mary Programming (Почему Мэри не программирует)?»

Если менеджер проекта претендует на роль бригадного генерала и приказывает вам немедленно начать программировать, вы можете с легкостью ответить: «Есть, сэр!» (И впрямь, какое вам дело? Умудренные опытом ветераны должны отвечать за свои слова.) Это плохой ответ, и у вас есть несколько лучших вариантов. Во-первых, вы можете решительно отвергнуть неэффективную методику работы. Если у вас нормальные отношения с начальником и все в порядке с банковским счетом, это может сработать.

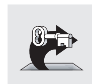
Во-вторых, вы можете притвориться, что работаете над кодом. Разложите на столе листинги старой программы и продолжайте работать над требованиями и архитектурой как ни в чем не бывало. Так вы выполните проект быстрее и качественнее. Порой этот подход находят неэтичным, но начальник-то останется доволен!

В-третьих, вы можете посвятить руководителя в нюансы технических проектов. Это хороший подход, потому что он увеличивает число грамотных руководителей в мире. В следующем подразделе приведено подробное обоснование важности выполнения предварительных условий до начала конструирования.

Наконец, вы можете найти другую работу. Независимо от экономических подъемов и спадов хороших программистов всегда не хватает (BLS, 2002), а жизнь слишком коротка, чтобы тратить ее на работу в отсталом учреждении при наличии множества лучших вариантов.

Самый веский аргумент в пользу выполнения предварительных условий перед началом конструирования

Допустим, вы уже забрались на гору определения проблемы, прошли милю по пути выработки требований, сбросили грязную одежду у фонтана архитектуры и искупались в чистых водах подготовленности. Следовательно, вы знаете, что перед реализацией системы нужно понимать, что и как она будет делать.



Один из аспектов профессии разработчика — посвящение профанов в особенности процесса разработки ПО. Этот раздел поможет вам в общении с менеджерами и руководителями, еще блуждающих во тьме. В нем подробно описан веский аргумент в пользу адекватного определения требований и проектирования архитектуры до начала кодирования, тестирования и отладки. Изучите его, сядьте перед начальником и поговорите о процессе программирования по душам.

Обращение к логике

Подготовка к проекту — одно из главных условий эффективного программирования, и это логично. Объем планирования зависит от масштаба проекта. С управленческой точки зрения, планирование подразумевает определение сроков, числа людей и компьютеров, необходимых для выполнения работ. С технической — планирование подразумевает получение представления о создаваемой системе, позволяющего не истратить деньги на создание неверной системы. Иногда пользователи не четко знают, что желают получить, и для определения их требований может понадобиться больше усилий, чем хотелось бы. Как бы то ни было, это дешевле, чем создать не то, что нужно, похерить результат и начать все заново.

До начала создания системы не менее важно подумать и о том, как вы собираетесь ее создавать. Никому не хочется тратить время и деньги на бесплодные блуждания по лабиринту.

Обращение к аналогии

Создание программной системы похоже на любой другой проект, требующий людских и финансовых ресурсов. Возведение дома начинается не с забивания гвоздей, а с создания, анализа и утверждения чертежей. При разработке ПО наличие технического плана означает не меньше.

Никто не наряжает новогоднюю елку, не установив ее. Никто не разводит огонь, не открыв дымоход. Никто не отправляется в долгий путь с пустым бензобаком. Никто не принимает душ в одежде и не надевает носки после обуви. И т. д., и т. п.

Программисты — последнее звено пищевой цепи разработки ПО. Архитекторы поглощают требования, проектировщики потребляют архитектуру, а программисты — проект приложения.

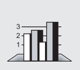
Сравните пищевую цепь разработки ПО с реальной пищевой цепью. В экологически чистой среде водные жучки служат пищей рыбам, которыми в свою очередь питаются чайки. Это здоровая пищевая цепь. Если на каждом этапе разработки ПО у вас будет здоровая пища, результатом станет здоровый код, написанный довольными программистами.

Если среда загрязнена, жучки плавают в ядерных отходах, а рыба плещется в нефтяных пятнах. Чайкам не повезло больше всего: находясь в конце пищевой цепи, они травятся и нефтью, и ядерными отходами. Если ваши требования неудачны, они отравляют архитектуру, которая в свою очередь травит процесс конструирования. Результат? Раздражительные программисты и полное изъянов ПО.

При планировании в высокой степени итеративного проекта вы до начала конструирования должны определить важнейшие требования и архитектурные элементы, влияющие на каждый конструируемый фрагмент программы. Строителям, собирающимся строить поселок, не нужна полная информация о каждом доме до начала возведения первого дома, однако они должны исследовать место, составить план канализации и электрических линий и т. д. Если строители плохо подготовятся, канализационные трубы, возможно, придется проводить в уже построенный дом.

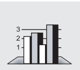
Обращение к данным

Исследования последних 25 лет убедительно доказали выгоду правильного выполнения проектов с первого раза и дороговизну внесения изменений, которых можно было избежать.



Ученые из компаний Hewlett-Packard, IBM, Hughes Aircraft, TRW и других организаций обнаружили, что исправление ошибки к началу конструирования обходится в 10–100 раз дешевле, чем ее устранение в конце работы над проектом, во время тестирования приложения или после его выпуска (Fagan, 1976; Humphrey, Snyder, and Willis, 1991; Leffingwell 1997; Willis et al., 1998; Grady, 1999; Shull et al., 2002; Boehm and Turner, 2004).

Общий принцип прост: исправлять ошибки нужно как можно раньше. Чем дольше дефект сохраняется в пищевой цепи разработки ПО, тем больше вреда он приносит на следующих этапах. Так как раньше всего вырабатываются требования, ошибки, допущенные на этом этапе, присутствуют в системе дольше и обходятся дороже. Кроме того, дефекты, внесенные в систему раньше, оказывают более широкое влияние, чем дефекты, внесенные позднее. Это также повышает цену более ранних дефектов.



Вот данные об относительной дороговизне исправления дефектов в зависимости от этапов их внесения и обнаружения (табл. 3-1):