Манифест гибкой разработки программ

Мы открываем лучшие способы разработки программного обеспечения, применяя их на практике и помогая в этом другим. В ходе своей работы мы научились ценить:

Людей и их взаимоотношения
больше, чем процессы и инструменты.

Работающую программу
больше, чем исчерпывающую документацию.

Плодотворное сотрудничество с заказчиком
больше, чем формальные договоренности по контракту.

Оперативное реагирование на изменения
больше, чем следование плану.

Иначе говоря, не отрицая значимость факторов, перечисленных во второй части предложений, мы больше ценим те, что в первой части.

Кент Бек
Майк Бидл
Ари ван Беннекум
Алистер Кокбэрн
Уорд Каннингэм
Мартин Фаулер
Джеймс Греннинг
Джим Хайсмит
Эндрю Хант
Рон Джеффрис
Йон Керн
Брайан Мэрик
Роберт К. Мартин
Стив Меллор
Кен Швабер
Джефф Сазерленд
Дэйв Томас

Принципы Манифеста гибкой разработки

Мы придерживаемся следующих принципов:

- Высшим приоритетом считать удовлетворение пожеланий заказчика посредством поставки полезного программного обеспечения в сжатые сроки с последующим непрерывным обновлением.
- Не игнорировать изменение требований, пусть даже на поздних этапах разработки. Гибкие процессы позволяют учитывать изменения и тем самым обеспечивать заказчику конкурентные преимущества.
- В процессе разработки предоставлять промежуточные версии ПО часто, с интервалом от пары недель до пары месяцев, отдавая предпочтение меньшим срокам.
- Заказчики и разработчики должны работать совместно на протяжении всего проекта.
- Проекты должны воплощать в жизнь целеустремленные люди. Создайте им условия, обеспечьте необходимую поддержку и верьте, что они доведут дело до конца.
- Самый эффективный и продуктивный метод передачи информации команде разработчиков и обмена мнениями внутри нее – разговор лицом к лицу.
- Работающая программа – основной показатель прогресса в проекте.
- Гибкие процессы способствуют долгосрочной разработке. Заказчики, разработчики и пользователи должны быть в состоянии поддерживать неизменный темп сколь угодно долго.
- Непрестанное внимание к техническому совершенству и качественному проектированию повышает отдачу от гибких технологий.
- Простота – искусство достигать большего, делая меньше, – основа основ.
- Самые лучшие архитектуры, требования и проекты выдают самоорганизующиеся команды.
- Команда должна регулярно задумываться над тем, как стать еще более эффективной, а затем соответственно корректировать и подстраивать свое поведение.

Методики экстремального программирования

Единая команда

Все участники XP-проекта, бизнес-аналитики, тестировщики и т. д. находятся в открытом рабочем помещении и являются членами одной команды. На стенах развешаны крупные, видные всем графики и другие свидетельства продвижения вперед.

Игра в планирование

Планирование непрерывно и поступательно. Каждые две недели разработчики оценивают стоимость функций, предложенных для реализации в течение следующих двух недель, а заказчик отбирает те функции, которые должны быть реализованы исходя из их стоимости и ценности для бизнеса.

Тесты пишет заказчик

При выборе каждой желаемой функции заказчик составляет автоматизированные приемочные тесты, демонстрирующие, что данная функция работает.

Простой дизайн

Команда разрабатывает такой дизайн, который отвечает текущей функциональности системы, – ни больше ни меньше. Он должен проходить все тесты, быть свободным от дублирования, выражать все, что хотел выразить автор, и содержать как можно меньше кода.

Парное программирование

Весь код пишется двумя программистами, сидящими бок о бок за одним компьютером.

Разработка через тестирование

Программисты работают очень короткими циклами, сначала добавляя тест, который не проходит, а затем код, при котором тест завершается успешно.

Улучшение дизайна

Плохой код не должен доживать до заката. Код все время должен оставаться максимально чистым и выразительным.

Непрерывная интеграция

Система все время остается интегрированной.

Коллективное владение кодом

Любая пара программистов вправе в любой момент усовершенствовать любой код.

Стандарты кодирования

Весь код выглядит так, будто его писал один – очень квалифицированный – человек.

Метафора

Команда поддерживает общее видение работы программы.

Умеренный темп

Команда собралась для длительной работы. Она работает усердно, но в таком темпе, который можно поддерживать сколь угодно долго. Команда бережет силы, рассматривая проект как марафон, а не спринт.

Принципы объектно-ориентированного проектирования

SRP	Принцип единственной обязанности *У класса должна быть только одна причина для изменения.*
OCP	Принцип открытости/закрытости *Программные сущности (классы, модули, функции и т. п.) должны быть открыты для расширения, но закрыты для модификации.*
LSP	Принцип подстановки Лисков *Должна быть возможность вместо базового типа подставить любой его подтип.*
DIP	Принцип инверсии зависимости *Абстракции не должны зависеть от деталей. Детали должны зависеть от абстракций.*
ISP	Принцип разделения интерфейсов *Клиенты не должны вынужденно зависеть от методов, которыми не пользуются. Интерфейсы принадлежат клиентам, а не иерархиям.*
REP	Принцип эквивалентности повторного использования и выпуска *Единица повторного использования равна единице выпуска.*
CCP	Принцип общей закрытости *Все классы внутри пакета должны быть закрыты относительно изменений одного и того же вида. Изменение, затрагивающее пакет, должно затрагивать все классы в этом пакете и только в нем.*
CRP	Принцип совместного повторного использования *Все классы внутри компонента используются совместно. Если вы можете повторно использовать один класс, то можете использовать и все остальные.*
ADP	Принцип ацикличности зависимостей *В графе зависимостей между пакетами не должно быть циклов.*
SDP	Принцип устойчивых зависимостей *Зависимости должны быть направлены в сторону устойчивости.*
SAP	Принцип устойчивых абстракций *Пакет должен быть столь же абстрактным, сколь и устойчивым.*

Оглавление

Предисловие 21
Об авторах 24
Введение 25

Часть I. Гибкая разработка 35

Глава 1. Гибкие методики 37
Организация Agile Alliance 38
Люди и их взаимоотношения
важнее процессов и инструментов 39
Работающая программа
важнее исчерпывающей документации 40
Плодотворное сотрудничество с заказчиком
важнее формальных договоренностей по контракту 41
Оперативное реагирование на изменения
важнее следования плану 42
Принципы 43
Заключение 45
Библиография 46

Глава 2. Обзор экстремального программирования 47
Методики экстремального программирования 48
Единая команда 48
Пользовательские истории 48
Короткие циклы 49
Приемочные тесты 50
Парное программирование 50
Разработка через тестирование 51
Коллективное владение 52
Непрерывная интеграция 52
Умеренный темп 53
Открытое рабочее пространство 53
Игра в планирование 53
Простота 54
Рефакторинг 55
Метафора 56

Заключение 57
Библиография 57
Глава 3. Планирование 58
Первичное обследование 59
Объединение, разбиение и скорость 59
Планирование выпуска 60
Планирование итерации 61
Определения понятия «готово» 61
Планирование задач 62
Подведение итогов итераций 63
Мониторинг 63
Заключение 64
Библиография 65
Глава 4. Тестирование 66
Разработка через тестирование 67
Пример проектирования начиная с тестов 68
Изоляция тестов 69
Разделение в придачу 71
Приемочные тесты 72
Архитектура в придачу 74
Заключение 74
Библиография 75
Глава 5. Рефакторинг 76
Простой пример рефакторинга: генерация простых чисел 77
Автономное тестирование 79
Рефакторинг 80
Последнее прочтение 85
Заключение 89
Библиография 90
Глава 6. Эпизод программирования 91
Игра в боулинг 93
Заключение 137
Часть II. Гибкое проектирование 139
Глава 7. Что такое гибкое проектирование 141
Ароматы дизайна 142
Ароматы дизайна – запахи гниющей программы 142
Жесткость 143
Хрупкость 143
Косность 143

Вязкость ... 143
Ненужная сложность ... 144
Ненужные повторения ... 144
Непрозрачность ... 145
Почему программы загнивают ... 145
Программа Copy ... 146
Знакомый сценарий ... 146
Гибкий дизайн программы Copy ... 150
Заключение ... 152
Библиография ... 152
Глава 8. Принцип единственной обязанности (SRP) ... 153
Определение обязанности ... 155
Разделение связанных обязанностей ... 157
Обеспечение сохранности ... 157
Заключение ... 158
Библиография ... 158
Глава 9. Принцип открытости/закрытости (OCP) ... 159
Описание принципа OCP ... 160
Приложение Shape ... 162
Нарушение OCP ... 163
Решение, удовлетворяющее принципу OCP ... 165
Предвидение и «естественная» структура ... 167
Расстановка «точек подключения» ... 168
Применение абстракции для явного закрытия ... 169
Закрытие за счет применения таблицы данных ... 170
Заключение ... 171
Библиография ... 171
Глава 10. Принцип подстановки Лисков (LSP) ... 172
Нарушения принципа LSP ... 173
Простой пример ... 173
Более тонкое нарушение ... 175
Реальный пример ... 181
Факторизация вместо наследования ... 186
Эвристика и соглашения ... 188
Заключение ... 189
Библиография ... 189
Глава 11. Принцип инверсии зависимости (DIP) ... 190
Разбиение на слои ... 191
Инверсия владения ... 192
Зависимость от абстракций ... 194

Простой пример принципа DIP 195
Отыскание глубинных абстракций 196
Задача об управлении печью 197
Заключение 199
Библиография 199
Глава 12. Принцип разделения интерфейсов (ISP) 200
Загрязнение интерфейса 200
Разделение клиентов означает разделение интерфейсов 202
Интерфейсы классов и интерфейсы объектов 204
Разделение путем делегирования 204
Разделение путем множественного наследования 205
Пользовательский интерфейс банкомата 206
Заключение 212
Библиография 213
Глава 13. Обзор UML для программистов 214
Диаграммы классов 218
Диаграммы объектов 219
Диаграммы последовательности 220
Диаграммы кооперации 220
Диаграммы состояний 221
Заключение 222
Библиография 222
Глава 14. Работа с диаграммами 223
Зачем нужно моделировать 223
Зачем строить модели программ 224
Нужно ли проектировать систему до конца, прежде чем приступать к кодированию 224
Эффективное использование UML 225
Общение с другими людьми 225
Карты 227
Финальная документация 228
Что сохранить, а что выбросить 229
Итеративное уточнение 230
Сначала поведение 230
Проверяем структуру 232
Набрасываем код 234
Эволюция диаграмм 235
Когда и как рисовать диаграммы 236
Когда рисовать диаграммы и когда остановиться 236
CASE-средства 237

Ну а как насчет документации? ... 238
Заключение ... 239
Глава 15. Диаграммы состояний ... 240
Основные понятия ... 241
Специальные события ... 242
Суперсостояния ... 243
Начальное и конечное псевдосостояния ... 245
Использование диаграмм состояний ... 245
Заключение ... 247
Глава 16. Диаграммы объектов ... 248
Мгновенный снимок ... 249
Активные объекты ... 250
Заключение ... 254
Глава 17. Прецеденты ... 255
Записывание прецедентов ... 256
Альтернативные потоки событий ... 257
Что-нибудь еще? ... 258
Представление прецедентов на диаграммах ... 258
Заключение ... 259
Библиография ... 259
Глава 18. Диаграммы последовательности ... 260
Основные понятия ... 261
Объекты, линии жизни, сообщения и прочее ... 261
Создание и уничтожение ... 262
Простые циклы ... 264
Различные сценарии ... 264
Более сложные конструкции ... 268
Циклы и условия ... 268
Сообщения, занимающие время ... 269
Асинхронные сообщения ... 270
Несколько потоков ... 276
Активные объекты ... 276
Отправка сообщений интерфейсам ... 277
Заключение ... 278
Глава 19. Диаграммы классов ... 280
Основные понятия ... 281
Классы ... 281
Ассоциация ... 282
Наследование ... 282
Пример диаграммы классов ... 284

Детали ... 286
Стереотипы классов ... 286
Абстрактные классы ... 288
Свойства ... 288
Агрегирование ... 289
Композиция ... 290
Кратность ... 292
Стереотипы ассоциаций ... 292
Вложенные классы ... 294
Классы ассоциаций ... 294
Квалификаторы ассоциаций ... 295
Заключение ... 295
Библиография ... 296
Глава 20. Эвристика и кофе ... 297
Кофеварка Mark IV Special ... 298
Спецификация ... 298
Типичное, но никуда не годное решение ... 301
Иллюзорная абстракция ... 303
Улучшенное решение ... 305
Реализация абстрактной модели ... 310
Достоинства описанного дизайна ... 317
Объектно-ориентированный перебор ... 318
Библиография ... 331
Часть III. Задача о расчете заработной платы ... 333
Краткая спецификация системы расчета заработной платы ... 334
Упражнение ... 335
Прецедент 1: добавление нового работника ... 335
Прецедент 2: удаление работника ... 336
Прецедент 3: регистрация карточки табельного учета ... 336
Прецедент 4: регистрация справки о продажах ... 336
Прецедент 5: регистрация платежного требования от профсоюза ... 336
Прецедент 6: изменение сведений о работнике ... 337
Прецедент 7: расчет заработной платы на сегодня ... 337
Глава 21. Команда и Активный объект: многогранность и многозадачность ... 338
Простые команды ... 339
Транзакции ... 341
Разрыв физических и темпоральных связей ... 343
Разрыв темпоральных связей ... 343

Метод Undo ... 344
Активный объект ... 345
Заключение ... 350
Библиография ... 350

Глава 22. Шаблонный метод и Стратегия: наследование или делегирование ... 351
Шаблонный метод ... 352
Злоупотребление паттерном ... 355
Пузырьковая сортировка ... 356
Стратегия ... 359
Заключение ... 364
Библиография ... 364

Глава 23. Фасад и Посредник ... 365
Фасад ... 365
Посредник ... 367
Заключение ... 369
Библиография ... 369

Глава 24. Одиночка и Моносостояние ... 370
Одиночка ... 371
Достоинства ... 373
Недостатки ... 373
Одиночка в действии ... 373
Моносостояние ... 375
Достоинства ... 376
Недостатки ... 377
Моносостояние в действии ... 377
Заключение ... 382
Библиография ... 383

Глава 25. Null-объект ... 384
Описание ... 384
Заключение ... 387
Библиография ... 387

Глава 26. Система расчета заработной платы: первая итерация ... 388
Краткая спецификация ... 389
Анализ по прецедентам ... 390
Добавление работников ... 391
Удаление работников ... 392
Регистрация карточки табельного учета ... 393
Регистрация справки о продажах ... 393

Регистрация платежного требования от профсоюза 394
Изменение сведений о работнике 395
Расчетный день 397
Осмысление:
поиск основополагающих абстракций 399
Тарификация работника 400
График выплат 400
Способы платежа 401
Принадлежность к другим организациям 402
Заключение 403
Библиография 403
Глава 27. Система расчета заработной платы: реализация 404
Операции 405
Добавление работников 405
Удаление работников 411
Карточки табельного учета,
справки о продажах и плата за услуги 413
Изменение сведений о работнике 420
Что я курил? 431
Начисление зарплаты 434
Начисление зарплаты работникам с твердым окладом 437
Начисление зарплаты работникам с почасовой оплатой 439
Головная программа 449
База данных 450
Заключение 452
Об этой главе 452
Библиография 453

Часть IV. Пакетирование системы расчета заработной платы 455

Глава 28. Принципы проектирования пакетов и компонентов 456
Пакеты и компоненты 457
Принципы сцепленности компонентов: детальность 458
Принцип эквивалентности повторного использования
и выпуска (Reuse/Release Equivalence Principle – REP) 458
Принцип общей закрытости
(Common Closure Principle – CCP) 461
Резюме 462
Принципы связанности компонентов: устойчивость 462
Принцип ацикличности зависимостей
(Acyclic Dependencies Principle – ADP) 462
Принцип устойчивых зависимостей
(Stable-Dependencies Principle – SDP) 469

Принцип устойчивых абстракций
(Stable-Abstractions Principle – SAP) 474
Заключение 479

Глава 29. Фабрика 480
Проблема зависимости 483
Статическая и динамическая типизация 484
Взаимозаменяемые фабрики 485
Использование фабрик в тестовых фикстурах 485
Важность фабрик 487
Заключение 488
Библиография 488

Глава 30. Система расчета заработной платы: анализ пакетов 489
Структура компонентов и обозначения 490
Применение принципа общей закрытости (CCP) 492
Применение принципа эквивалентности
повторного использования и выпуска (REP) 494
Связанность и инкапсуляция 497
Метрики 498
Применение метрик к задаче о расчете зарплаты 500
Фабрики объектов 504
Еще раз о границах сцепленности 506
Окончательная структура пакетов 507
Заключение 510
Библиография 510

Глава 31. Компоновщик 511
Составные команды 513
Кратный или не кратный 514
Заключение 514

Глава 32. Наблюдатель: превращение в паттерн 515
Цифровые часы 516
Паттерн Наблюдатель 535
Модели 535
Связь с принципами ООП 536
Заключение 537
Библиография 538

Глава 33. Абстрактный сервер, адаптер и мост 539
Абстрактный сервер 540
Адаптер 542
Адаптер класса 543
Задача о модеме, адаптеры и принцип LSP 543
Мост 547

Заключение 550
Библиография 550

Глава 34. Заместитель и Шлюз: управление сторонними API 551
Заместитель 552
Реализация Заместителя 557
Резюме 570
Базы данных, ПО промежуточного уровня и прочие сторонние интерфейсы 571
Шлюз к табличным данным 573
Тестирование и шлюз к табличным данным в памяти 580
Тестирование шлюзов к базе данных 581
Другие паттерны работы с базами данных 584
Заключение 586
Блиблиография 586

Глава 35. Посетитель 587
Посетитель 588
Ациклический посетитель 592
Применения паттерна Посетитель 597
Декоратор 604
Объект расширения 610
Заключение 621
Библиография 621

Глава 36. Состояние 622
Вложенные предложения switch/case 623
Поле State с внутренним доступом 626
Тестирование действий 626
Достоинства и недостатки 627
Таблицы переходов 627
Интерпретация таблицы 628
Достоинства и недостатки 629
Паттерн Состояние 630
Паттерны Состояние и Стратегия 633
Достоинства и недостатки 634
Компилятор конечных автоматов (SMC) 634
Файл Turnstile.cs, сгенерированный SMC, и другие вспомогательные файлы 637
Виды приложений конечных автоматов 642
Высокоуровневые политики приложения и ГИП 642
Контроллеры взаимодействия с ГИП 644
Распределенная обработка 645

Заключение 646
Библиография 646
Глава 37. Система расчета заработной платы: база данных 647
Построение базы данных 648
Изъян в дизайне программы 649
Добавление работника 651
Транзакции 662
Выборка работника 668
Что осталось? 681
Глава 38. Система расчета заработной платы: Модель-Вид-Презентатор 682
Интерфейс 684
Реализация 686
Конструирование окна 696
Окно Payroll 703
Снимаем покрывало 715
Заключение 716
Библиография 716
Приложение A. Сказ о двух компаниях 717
Приложение B. Что такое проектирование программного обеспечения 734
Алфавитный указатель 748

Предисловие

От Криса Селлса

Моим дебютом на поприще профессионального программирования стало добавление некоторых функций в базу данных о вредителях – тле, саранче, гусеницах – по заказу отдела патологии растений экспериментальных ферм университета штата Миннесота. Код был написан энтомологом, который выучил dBase ровно настолько, чтобы суметь написать одну форму, а потом методом копирования распространить ее на все приложение. Расширяя функциональность приложения, я по мере возможности объединял различные функции, так чтобы можно было исправлять ошибки, внося изменения лишь в одном месте, улучшения тоже производить в одном месте и т. д. Это заняло у меня все лето, но в итоге я вдвое расширил функциональность приложения, одновременно вдвое уменьшив объем кода.

Спустя много-много лет у нас с приятелем выдался период, когда не было срочной работы, и мы решили вместе написать какую-нибудь программу (если быть точным, реализацию одного из интерфейсов IDispatch или IMoniker, которые в то время занимали все наши мысли). Сначала я набирал код, а он стоял сзади и указывал мне на ошибки. Потом мы менялись местами, и я поучал его, пока он не возвращал управление мне.

Так продолжалось часами, и лучшего опыта в моей практике программирования не было. Вскоре после этого приятель предложил мне занять место главного архитектора во вновь образованном отделе разработки ПО в его компании. Работая архитектором, я нередко писал клиентский код для еще не существовавших объектов и передавал его инженерам, которые реализовывали объекты так, чтобы этот код заработал.

Полагаю, что мои эксперименты с различными аспектами гибкой разработки не были уникальными, как не были уникальными занятия подростков, изучавших различные любовные техники на заднем сиденье Шевроле 1957 года в те времена, когда половое воспитание еще не стало частью стандартного учебного плана. В общем и целом, мои опыты с такими методами гибкой разработки, как рефакторинг, парное программирование и разработка через тестирование, оказались успешными, хотя я еще не знал, что именно делаю. Конечно, и до выхода этой книги были материалы по гибким методикам, но, образно говоря,

мне совсем не хочется осваивать народные танцы по старым номерам National Geographic. Технологии гибкой разработки интересуют меня лишь применительно к платформе .NET, с которой работает моя группа. Рассказывая о .NET, Роберт (хотя он ясно дает понять, что во многих случаях .NET ничем не лучше Java) говорит на моем языке, как те преподаватели старших классов, которые давали себе труд изучать слэнг подростков, понимая, что сообщение важнее носителя.

Но дело не только в .NET; я хотел бы, чтобы начало изучения было ненавязчивым, неторопливым, не отпугивало меня, но чтобы при этом я узнал все то, что знать необходимо. Именно так Роберт «Дядя Боб» Мартин и организовал эту книгу. В первых главах основы гибкой разработки излагаются постепенно, не заставляя читателя очертя голову бросаться в омут SCRUM, экстремального программирования и других конкретных методик, а давая возможность атомам собираться в такие молекулы, где они чувствуют себя наиболее комфортно. Но еще больше мне нравится в стиле Роберта то, как он демонстрирует описываемые приемы в действии. Взяв некоторую реальную задачу, он показывает, какие ошибки и ложные ходы можно допустить в ходе ее решения и как применение методов, за которые он ратует, позволяет вернуться на твердую почву.

Не знаю, существует ли в реальности мир, описываемый Робертом в этой книге; за всю мою жизнь мне удалось лишь мельком взглянуть на него. Но совершенно ясно, что все «крутые ребята» живут именно в нем. Считайте «дядю Боба» своим личным «доктором Рут»[1] в мире гибкой разработки, чья единственная цель – научить вас выполнять эту работу хорошо, чтобы все чувствовали удовлетворенность своей деятельностью.

Крис Селлс

От Эриха Гаммы

Я пишу это предисловие сразу после выпуска очередной основной версии проекта Eclipse с открытым исходным кодом. Я еще не оправился от напряженного труда, и мозг немного затуманен. Но одну вещь я понимаю яснее, чем когда-либо: ключ к своевременной поставке продукта – не процессы, а люди. Наш рецепт успеха прост: работать с людьми, по-настоящему увлеченными программированием, применять в разработке упрощенные процессы, подстроенные под конкретную команду, и непрестанно адаптироваться.

Если «дважды щелкнуть» по любому разработчику из наших команд, то обнаружится личность, считающая программирование центром разработки. Они не просто пишут код, а еще и постоянно продумывают его с позиций понимания работы системы в целом. Проверка проекта с по-

[1] Рут Вестхаймер, американский сексопатолог, ведущая теле- и радиопередач, автор множества книг. – *Прим. перев.*

мощью кода дает обратную связь, без которой невозможно получить уверенность в отсутствии ошибок проектирования. Но в то же время наши разработчики понимают важность паттернов, рефакторинга, тестирования, инкрементной поставки, частого выполнения сборки и прочих методик экстремального программирования, изменивших наш взгляд на методологию разработки ПО.

Навыки в таком стиле разработки – необходимое условие успеха в проектах с высоким техническим риском и изменяющимися требованиями. Гибкая разработка молчит, когда речь идет о формальностях и проектной документации, но возвышает голос, когда дело доходит до повседневных практических методик, которые действительно имеют значение. Собрать эти методики воедино и заставить их работать – вот цель данной книги.

Роберт уже давно является активным членом сообщества объектно-ориентированного программирования, внесшим немалый вклад в практическое применение C++, паттерны проектирования и принципы объектно-ориентированной разработки в целом. Он с самого начала активно отстаивал методы экстремального программирования и гибкой разработки. Эта книга, основанная на его богатом опыте, охватывает все аспекты практического применения гибкой разработки. Амбициозная задача, ничего не скажешь. Решая ее, Роберт приводит разнообразные примеры, сопровождая их кодом, как и подобает адепту гибкой разработки. Он учит программированию и проектированию на практике.

Эта книга буквально напичкана мудрыми советами, как разрабатывать ПО. Она с равным успехом подойдет и тому, кто еще только собирается практиковать гибкую разработку, и тому, кто желает усовершенствовать уже имеющиеся навыки. Я с нетерпением ждал ее выхода и не был разочарован.

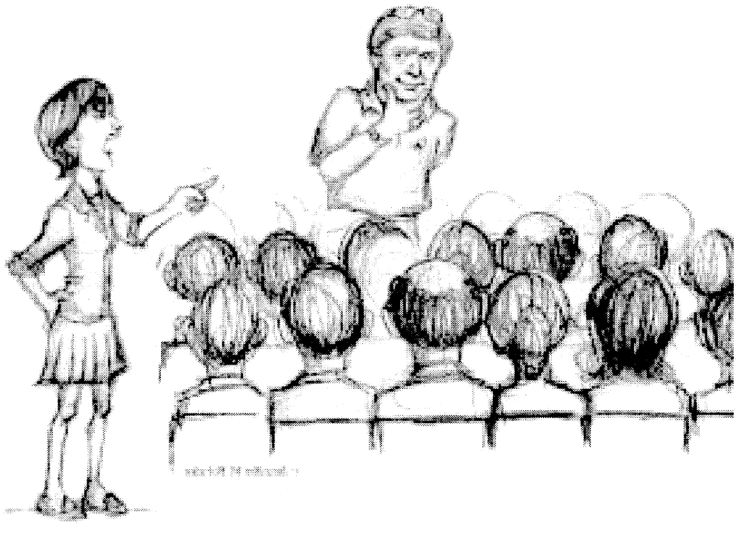
Эрих Гамма,
Object Technology International

Об авторах

Роберт К. Мартин («Дядя Боб») – основатель и президент международной компании Object Mentor Inc. со штаб-квартирой в Гурни, штат Иллинойс, предлагающей консультативные услуги по совершенствованию процесса разработки, объектно-ориентированному проектированию, обучению и повышению квалификации разработчиков крупным компаниям по всему миру. Он автор книг «Designing Object Oriented C++ Applications Using the Booch Method» и «Agile Software Development Principles, Patterns, and Practices» (обе вышли в издательстве Prentice Hall), а также «UML for Java Programming» (Addison-Wesley). В период с 1996 по 1999 год был главным редактором журнала «C++ Journal». Известен своими выступлениями на международных конференциях и промышленных выставках.

Мика Мартин трудится в компании Object Mentor в качестве разработчика, консультанта и наставника по различным предметам, начиная от объектно-ориентированных принципов и паттернов и кончая методиками гибкой разработки ПО. Мика – сооснователь и ведущий разработчик проекта FitNesse с открытым исходным кодом. Он также автор ряда печатных работ, регулярно выступает на конференциях.

Введение



Но Боб, ты же говорил, что закончишь книгу в ***прошлом*** *году.*

Клаудиа Фрерс, UML World, 1999

От Боба

Прошло уже семь лет с момента заявления этой справедливой претензии Клаудии, и думается, что я искупил свою вину. Опубликовать *три* книги – по одной каждые два года – и при этом руководить консалтинговой фирмой, писать уйму кода, преподавать, обучать, выступать на конференциях, писать статьи, вести блоги и колонки в разных журналах, не говоря уже об обязанностях по отношению к своей семье и родителям, – это нелегко. Но мне нравится такая жизнь.

Гибкая разработка – это умение быстро разрабатывать ПО в условиях постоянно изменяющихся требований. Чтобы достичь такой гибкости, необходимо освоить методики, диктующие определенную дисциплину и установление обратной связи. Мы должны применять принципы проектирования, благодаря которым программы будут оставаться гибки-

ми и удобными для сопровождения. Нам также нужны проверенные практикой паттерны проектирования, которые позволяют применить эти принципы к конкретным проблемам. Эта книга – попытка связать все три концепции в единое работающее целое.

Сначала описываются принципы, паттерны и методики, а затем на десятках различных примеров демонстрируется их применение. Важнее, однако, что примеры представлены в виде не завершенных работ, а разворачивающегося *процесса проектирования*. Вы увидите, что проектировщики тоже совершают ошибки, станете свидетелями того, как они выявляют эти ошибки и в конечном итоге исправляют их. Вы увидите, как проектировщики бьются над головоломными задачами, как они добиваются ясности и определенности и как вырабатывают компромиссы. Вы увидите само *действо* проектирования.

От Мики

В начале 2005 года я был членом небольшой группы разработчиков, которая начала работать над приложением .NET на языке C#. Обязательным условием было применение гибких методик разработки, в частности поэтому меня и пригласили. Хотя мне доводилось раньше писать на C#, лучше всего я владел языками Java и C++. Я не думал, что работа на платформе .NET будет чем-то существенно отличаться; так оно и оказалось.

Через два месяца работы над проектом была выпущена первая версия. Она содержала лишь часть запланированных функций, но при этом была в достаточной мере работоспособной. И с ней таки работали. Спустя всего два месяца организация-заказчик уже пожинала плоды наших трудов. Руководство было настолько поражено, что попросило нанять еще людей, чтобы мы могли приступить к новым проектам.

Принимая на протяжении многих лет участие в жизни сообщества гибкой разработки, я был знаком со многими приверженцами этой технологии, которые могли бы нам помочь. Я предложил им присоединиться к проекту. Но ни один из моих знакомых так и не влился в команду. Почему? Быть может, по той простой причине, что мы вели разработку на платформе .NET.

Почти все практики гибкой разработки имели опыт работы на Java, C++ или Smalltalk. Но о гибкой разработке в .NET тогда почти не слыхали. Возможно, мои друзья не приняли всерьез мои слова о том, что мы занимаемся гибкой разработкой в среде .NET, а быть может, не захотели связываться с .NET. Это оказалось серьезной проблемой. И я столкнулся с ней не впервые.

Проведение недельных курсов по различным аспектам программирования позволяет мне встречаться с самыми разными разработчиками по всему миру. Многие мои слушатели работали с .NET, но не меньше было

и программистов на Java или C++. И буду откровенен: мой опыт показывает, что программисты .NET зачастую слабее тех, что пишут на Java или C++. Понятно, что бывают исключения. Однако раз за разом наблюдая за своими слушателями, я вынужден был заключить, что программисты .NET обычно хуже разбираются в методах разработки ПО, паттернах и принципах проектирования и т. п. Нередко случалось, что присутствовавшие программисты .NET вообще не слыхали об этих фундаментальных концепциях. *Такое положение должно быть изменено.*

Первое издание этой книги, «Agile Software Development: Principles, Patterns, and Practices», написанной моим отцом, Робертом К. Мартином, вышло в 2002 году и получило премию Jolt Award 2003 года. Это замечательная книга, восторженно принятая многими разработчиками. К сожалению, она не оказала большого влияния на сообщество разработчиков в среде .NET. Хотя изложенные в книге идеи вполне применимы к .NET, немногие программирующие на этой платформе прочли ее.

Я надеюсь, что это издание, ориентированное на .NET, перекинет мост между разработчиками .NET и остальным сообществом. Я надеюсь, что программисты прочтут ее и увидят, что существуют лучшие способы построения программ. Я надеюсь, что они начнут использовать лучшие методики, создавать лучшие проекты и поднимут планку качества при разработке .NET-приложений. Я надеюсь, что программисты .NET завоюют новый статус в сообществе разработчиков, так что программисты на Java смогут с гордостью присоединиться к команде, работающей на платформе .NET.

Работая над этой книгой, я не раз сомневался, ставить ли свое имя на обложке книги, посвященной .NET. Я спрашивал себя, хочу ли я, чтобы меня ассоциировали с .NET и со всеми негативными представлениями, связанными с этой платформой. Но что толку отрицать? Я программист .NET. Нет! Я гибкий программист .NET. И горжусь этим.

Об этой книге

Немного истории

В начале 1990-х годов я (Боб) написал книгу «Designing Object-Oriented C++ Applications Using the Booch Method». Для меня она стала чем-то вроде magnum opus[1], и я был очень доволен результатом и тем, как книга продавалась.

Книга, которую вы сейчас держите в руках, была задумана как второе издание «Designing», но все обернулось иначе. От оригинальной книги осталось очень мало. В новую перешло чуть меньше трех глав, да и этот материал подвергся существенной переработке. Цель книги, характер

[1] Труд всей жизни. – *Прим. перев.*

изложения и многие уроки остались прежними. Но за десять лет, прошедших с момента выхода «Designing», я узнал очень много нового о проектировании и разработке ПО. И этот новый опыт нашел отражение в книге.

Ах, что это было за десятилетие! Книга «Designing» вышла еще до того, как Интернет покорил всю планету. С тех пор количество акронимов, с которыми мы сталкиваемся, удвоилось. EJB, RMI, J2EE, XML, XSLT, HTML, ASP, JSP, ZOPE, SOAP, C# и .NET... А к ним в придачу паттерны проектирования, Java, сервлеты и серверы приложений. Поверьте, наполнить главы этой книги актуальным материалом было непросто.

Сотрудничество с Бучем. В 1997 году Гради Буч попросил меня помочь ему в работе над третьим изданием его поразительно успешной книги «Object-Oriented Analysis and Design with Applications»[1]. Я и раньше работал с Гради над некоторыми проектами и с удовольствием читал его труды, в частности о UML, и вносил в них свой посильный вклад. Поэтому я с восторгом согласился и попросил своего старого друга Джима Ньюкирка присоединиться к этому проекту.

В последующие два года мы с Джимом написали несколько глав для книги Буча. Конечно, из-за этого я не смог уделять своей книге столько времени, сколько хотел бы, но мне казалось, что книга Буча стоит затраченных усилий. К тому же в то время эта книга представлялась мне просто вторым изданием «Designing», и душа у меня к ней не лежала. Уж если что-то говорить, то новое и незатертое.

К сожалению, книга Буча так и не вышла. Трудно найти время для написания книги без отрыва от основной работы. А уж в те годы стремительного развития интернет-компаний это было почти невозможно. Гради был безумно занят компанией Rational и другими венчурными предприятиями, такими как Catapulse. Поэтому проект застопорился. Я испросил у Буча и издательства Addison-Wesley разрешение включить написанные мною и Джимом главы в эту книгу. Они любезно согласились. Так появились некоторые примеры и главы, посвященные UML.

Влияние экстремального программирования. В конце 1998 года возникло экстремальное программирование (XP) и тут же бросило вызов нашим устоявшимся взглядам на разработку программного обеспечения. Нужно ли рисовать кучу UML-диаграмм перед тем, как приступить к написанию кода? Или лучше отказаться от всех диаграмм и просто писать код, много кода? Нужно ли составлять объемную документацию, в которой описывается проект? Или лучше сделать сам *код* настолько самодокументированным и ясным, что никакие дополнительные документы не понадобятся? Нужно ли программировать

[1] **Гради Буч «Объектно-ориентированный анализ и проектирование с примерами приложений на C++». – Пер. с англ. – Бином, Невский Диалект, 1998.**

вдвоем? Нужно ли писать тесты еще до написания основного кода? Что вообще нужно делать?

Для этой революции был выбран самый подходящий момент. С середины и до конца 1990-х годов фирма Object Mentor консультировала различные компании по вопросам объектно-ориентированного проектирования и управления проектами. Мы помогали реализовывать проекты, в частности внедряли в коллективы разработчиков свои представления и методы. К сожалению, они не были сформулированы в письменном виде. Это была просто устная традиция, передававшаяся от нас заказчикам.

К 1998 году я осознал, что необходимо описать наши процессы и методики, чтобы было проще излагать их заказчикам. Тогда я написал много статей в журнал *C++ Report*[1]. Но это был выстрел мимо цели. Статьи были информативными и даже иногда занимательными, но вместо того чтобы систематизировать те представления и методы, которые использовались нами в проектах, они оказались невольным компромиссом с теми концепциями, которые были навязаны мне за последние десятилетия. Указал мне на это Кент Бек.

Сотрудничество с Беком. В конце 1998 года, как раз тогда, когда я занимался систематизацией процесса разработки, принятого в компании Object Mentor, я наткнулся на работу Кента по экстремальному программированию (XP). Я нашел ее на сайте википедии[2] Уорда Каннингэма среди статей многих других авторов. И немного потрудившись, мне удалось вычленить основную мысль Кента. Я был заинтригован, но отнесся к ней скептически. Кое-какие положения XP идеально ложились на мою концепцию процесса разработки. Но некоторые моменты, например отсутствие четких шагов проектирования, озадачивали.

Вероятно, дело в том, что мы с Кентом работали в разных условиях. Он был известным консультантом по языку Smalltalk, я – не менее известным консультантом по C++. Общение между этими двумя мирами затруднено. Между обеими парадигмами пролегает почти кунианская[3] пропасть.

---

[1] Эти статьи можно найти в разделе *Publications* на сайте www.objectmentor.com. Всего их четыре. Первые три озаглавлены «Iterative and Incremental Development» (I, II и III), последняя – «C.O.D.E Culled Object Development process».

[2] На сайте *http://c2.com/cgi/wiki* размещено множество статей на самые разные темы. Количество авторов исчисляется сотнями или тысячами. Говорили, что только Уорд Каннингэм способен вызвать социальную революцию, написав всего несколько строк на Perl.

[3] Термин *«кунианский»* должен был присутствовать в каждом значимом интеллектуальном труде, написанном между 1995 и 2001 годами. Он восходит к книге Томаса С. Куна (Thomas S. Kuhn) «The Structure of Scientific Revolutions», издательство University of Chicago Press, 1962.

При таких обстоятельствах я никогда не попросил бы Кента написать статью в журнал *C++ Report*. Но схожесть наших размышлений о процессе разработки помогла преодолеть языковую пропасть. В феврале 1999 года я встретился с Кентом в Мюнхене на конференции по ООП. Он рассказывал об XP в аудитории, расположенной как раз напротив той, где я докладывал о принципах объектно-ориентированной разработки. Поскольку послушать его доклад мне не удалось, то я разыскал Кента во время обеденного перерыва. Мы поговорили об XP, и я предложил ему написать статью в *C++ Report*. Это была блистательная статья о реальном случае, когда Кенту вместе с коллегой удалось внести кардинальное изменение в проект работающей системы, затратив на это около часа.

На протяжении нескольких следующих месяцев я медленно избавлялся от собственных страхов по поводу XP. Самым пугающим мне представлялось отсутствие явного этапа проектирования, предшествующего всему остальному. С этим я никак не мог примириться. Разве не считал я всегда своей обязанностью перед заказчиками и индустрией в целом разъяснять, что проектирование – вещь достаточно важная, чтобы тратить на нее время?

Но в конце концов я осознал, что и сам-то так не поступаю. Да, во всех своих статьях и книгах я писал о проектировании, о диаграммах Буча и UML-диаграммах, но при этом всегда использовал код как средство проверки осмысленности диаграмм. Консультируя своих заказчиков, я тратил час-другой на то, чтобы помочь им нарисовать диаграммы, а потом показывал, как наполнить их кодом. Я начал понимать, что, хотя фразеология XP в части проектирования выглядела для меня чуждой в кунианском смысле[1], стоящая за словами практика была мне знакома.

С другими опасениями, касающимися XP, справиться оказалось легче. Я всегда был тайным любителем парного программирования. XP дало мне возможность сбросить покров тайны и открыто заявить о своем желании программировать вместе с партнером. Рефакторинг, непрерывная интеграция, работа совместно с заказчиком – мне было нетрудно принять все это, поскольку такие идеи были очень близки к тому, что я и сам советовал своим клиентам.

Одна из методик XP стала для меня откровением. Идея разработки через тестирование (Test-driven development – TDD)[2] на первый взгляд

---

[1] Если имя Куна упомянуто в статье дважды, вы получаете дополнительные очки.

[2] Kent Beck «Test-Driven Development by Example», Addison-Wesley, 2003. (Кент Бек «Экстремальное программирование: разработка через тестирование». – Пер. с англ. – Питер, 2003.)

звучит совершенно безобидно: пишите тесты перед созданием основного кода. А основной код следует писать так, чтобы ранее не проходившие тесты завершались успешно. Эта идея полностью изменила мой подход к написанию программ – и изменила в лучшую сторону.

Таким образом, к осени 1999 года я пришел к убеждению, что компания Object Mentor должна принять XP в качестве основного процесса разработки и что мне следует отказаться от желания описать свой собственный процесс. Кент уже проделал отличную работу по облечению в слова методики и процесса XP, мои робкие потуги бледнели перед этим трудом.

.NET. Между крупнейшими корпорациями идет война. Корпорации борются за то, чтобы завоевать *ваши* симпатии. Корпорации полагают, что, владея неким языком, они владеют и программистами, пишущими на этом языке, и компаниями, в которых эти программисты работают.

Первым залпом в этой битве стал язык Java. Этот язык был специально создан крупной корпорацией с намерением завоевать умы программистов. Его успех оказался ошеломительным. Java глубоко проник в программистское сообщество и стал стандартом де факто для разработки современных многоуровневых ИТ-приложений.

Ответный залп последовал от корпорации IBM, которая благодаря среде разработки Eclipse отхватила значительный сегмент рынка Java. Открыли огонь и непревзойденные стратеги в корпорации Microsoft, которые подарили нам платформу .NET вообще и язык C# в частности.

Забавно, что провести различие между Java и C# весьма затруднительно. Оба языка семантически эквивалентны, а синтаксически настолько схожи, что многие фрагменты кода выглядят совершенно одинаково. Но если Microsoft и отстает в плане технической новизны, то с лихвой компенсирует это удивительной способностью играть в «догонялки» и выигрывать.

В первом издании этой книги в качестве языков программирования использовались Java и C++. А это издание ориентировано на язык C# и платформу .NET. Но не следует считать это сменой пристрастий. Мы не примыкаем ни к одной из воюющих сторон. Вообще, я полагаю, что война затихнет сама собой, когда через несколько лет появится какой-нибудь лучший язык и привлечет на свою сторону программистов, которых враждующие корпорации так старались удержать.

А причина появления версии этой книги для .NET в том, чтобы охватить аудиторию, работающую на этой платформе. Хотя сами описываемые принципы, паттерны и методики от языка не зависят, сказать то же самое о примерах уже нельзя. Программистам .NET привычнее читать код примеров на .NET-совместимом языке, а программистам Java – код на Java.

Дьявол кроется в деталях

В этой книге приведено *очень много* кода. Мы надеемся, что вы будете читать его внимательно, потому что в значительной степени именно код и является *сутью* данной книги. Именно в коде воплощено все, что мы хотели передать вам в этой книге.

Книга представляет собой последовательность примеров разного объема. Некоторые совсем небольшие, другие занимают несколько глав. Каждому примеру предшествует материал, имеющий целью подготовить читателя: описание принципов и паттернов объектно-ориентированного проектирования, используемых в данном примере.

Книга начинается с обсуждения методик и процессов разработки. Это обсуждение завершается рядом небольших примеров. Далее мы переходим к теме проектирования и его принципов, затем – к некоторым паттернам, потом снова к принципам – управления пакетами, и опять к паттернам. Изложение неизменно сопровождается конкретными примерами.

Поэтому приготовьтесь читать код и разбираться в UML-диаграммах. Книга, на которую вы сейчас смотрите, содержит *преимущественно* техническую информацию, и ее уроки, как и дьявол, кроются в деталях.

Структура книги

Книга состоит из четырех разделов и двух приложений.

В части I «Гибкая разработка» описывается идея гибкой разработки. Она начинается с *Манифеста гибкой разработки*, затем дается обзор экстремального программирования (XP), после чего на ряде небольших примеров иллюстрируются некоторые приемы XP, в особенности те, что влияют на способы проектирования и написания кода.

В части II «Гибкое проектирование» речь пойдет об объектно-ориентированном проектировании ПО: что это такое, постановка задачи об управлении сложностью и методы ее решения, принципы объектно-ориентированного проектирования классов. Завершается эта часть несколькими главами, посвященными описанию использования подмножества UML на практике.

В части III «Задача о расчете заработной платы» описывается объектно-ориентированный проект и реализация на C# простой пакетной системы расчета заработной платы. В начальных главах мы рассказываем о паттернах проектирования, встречающихся в этом примере. А последнюю главу занимает полный пример – самый большой и сложный в этой книге.

Часть IV «Пакетирование системы расчета заработной платы» начинается с описания *принципов проектирования объектно-ориентированных пакетов*, после чего мы переходим к иллюстрации этих принципов на примере постепенной компоновки в пакеты классов из предыдущего раздела. Завершается часть главами, касающимися проектирования базы данных и пользовательского интерфейса для приложения «Система расчета заработной платы».

В книге есть два приложения: «Сказ о двух компаниях» и статья Джека Ривза «Что такое проектирование программного обеспечения».

Как читать эту книгу

Если вы разработчик, то читайте книгу от корки до корки. Она написана преимущественно для разработчиков и содержит информацию о том, как писать программы, применяя гибкие методики. Читая книгу последовательно, вы сначала ознакомитесь с методиками, затем с принципами, с паттернами и наконец с примерами, где все это увязано воедино. Усвоение всех этих знаний поможет вам довести проект до успешного завершения.

Если вы менеджер или бизнес-аналитик, то прочитайте часть I «Гибкая разработка». В главах 1–6 вы найдете углубленное обсуждение принципов и методик гибкой разработки: от анализа требований до планирования, тестирования, рефакторинга и программирования. В части I приведены рекомендации по созданию команд и управлению проектами. Они помогут вам довести проект до успешного завершения.

Если вы хотите изучить UML, то начните с глав 13–19. Затем прочитайте все главы части III «Задача о расчете заработной платы». При такой последовательности чтения вы получите основательные знания о синтаксисе и использовании UML и сумеете перевести проект с языка диаграмм UML на C#.

Если вы хотите узнать о паттернах проектирования, прочитайте сначала материал о принципах проектирования в части II «Гибкое проектирование», а затем часть III «Задача о расчете заработной платы» и часть IV «Пакетирование системы расчета заработной платы». В них описаны все паттерны и показано, как применять их в типичных ситуациях.

Если вы хотите узнать о принципах объектно-ориентированного проектирования, прочитайте часть II «Гибкое проектирование», часть III «Задача о расчете заработной платы» и часть IV «Пакетирование системы расчета заработной платы». В них описаны принципы объектно-ориентированного проектирования и показано, как применять их на практике.

Если вы хотите узнать о методах гибкой разработки, прочитайте часть I «Гибкая разработка». Здесь описана методология гибкой разра-

ботки от анализа требований до планирования, тестирования, рефакторинга и программирования.

Если вы хотите немного развлечься, то прочитайте приложение А «Сказ о двух компаниях».

Благодарности

Мы благодарны Лоуэллу Линдстрому (Lowell Lindstrom), Брайану Баттону (Brian Button), Эрику Миду (Erik Meade), Майку Хиллу (Mike Hill), Майклу Фэзерсу (Michael Feathers), Джиму Ньюкирку (Jim Newkirk), Мике Мартину (Micah Martin), Анжелике Мартин (Angelique Martin), Сьюзен Россо (Susan Rosso), Талише Джефферсон (Talisha Jefferson), Рону Джеффрису (Ron Jeffries), Кенту Беку (Kent Beck), Джеффу Лэнгру (Jeff Langr), Дэвиду Фарберу (David Farber), Бобу Коссу (Bob Koss), Джеймсу Греннингу (James Grenning), Лансу Уэлтеру (Lance Welter), Паскалю Рою (Pascal Roy), Мартину Фаулеру (Martin Fowler), Джону Гудсену (John Goodsen), Алану Эпту (Alan Apt), Полу Ходжеттсу (Paul Hodgetts), Филу Маркграфу (Phil Markgraf), Питу Макбрину (Pete McBreen), Х. С. Леману (H. S. Lahman), Дэйву Харрису (Dave Harris), Джеймсу Канзе (James Kanze), Марку Уэбстеру (Mark Webster), Крису Бигею (Chris Biegay), Алану Фрэнсису (Alan Francis), Джессике Д'Амико (Jessica D'Amico), Крису Гузиковски (Chris Guzikowski), Полу Петралиа (Paul Petralia), Мишель Хаусли (Michelle Housley), Дэвиду Челимски (David Chelimsky), Полу Пагелю (Paul Pagel), Тиму Оттингеру (Tim Ottinger), Кристоферу Хэдгейту (Christoffer Hedgate) и Нилу Рудину (Neil Roodyn).

Особая благодарность Гради Бучу и Полу Беккеру за разрешение включить главы, первоначально предназначавшиеся для третьего издания книги Гради *«Object-Oriented Analysis and Design with Applications»*. Также особая благодарность Джеку Ривзу за любезное разрешение воспроизвести текст его статьи «Что такое проектирование программного обеспечения».

Прекрасные, а иногда просто восхитительные иллюстрации принадлежат Дженнифер Конке (Jennifer Kohnke) и моей дочери Анджеле Брукс (Angela Brooks).