

# СОДЕРЖАНИЕ

Об авторах	15
<b>Предисловие</b>	16
<b>Глава 1. Философия прагматизма</b>	29
<b>Тема 1. Это ваша жизнь</b>	30
<b>Тема 2. Кот съел мой исходный код</b>	31
Доверие в команде	32
Взятие на себя ответственности	32
<b>Тема 3. Программная энтропия</b>	34
Прежде всего — не навредить	36
<b>Тема 4. Суп из камней и вареные лягушки</b>	37
Со стороны селян	38
<b>Тема 5. Достаточно хорошее программное обеспечение</b>	40
Идите на компромиссы с пользователями	41
Знайте меру	42
<b>Тема 6. Ваш багаж знаний</b>	43
Ваш багаж знаний	43
Создание своего багажа знаний	44
Цели	45
Возможности для обучения	47
Критическое мышление	47
<b>Тема 7. Общайтесь!</b>	49
Знайте, с кем вы общаетесь	50
Знайте, что вам требуется сказать	51
Выбирайте удобный момент	51
Выбирайте стиль общения	52
Подавайте свои идеи в привлекательной форме	52
Привлекайте тех, с кем общаетесь	53
Учитесь слушать	53
Отвечайте людям	53
Документация	54
Краткие итоги	54
<b>Глава 2. Прагматичный подход</b>	57
<b>Тема 8. Сущность качественного проектирования</b>	58
Принцип ETC — это ценность, а не правило	58
<b>Тема 9. DRY — пороки дублирования</b>	60
Принцип DRY не только для кодирования	61
Дублирование в исходном коде	62

Дублирование в документации	64
Представительное дублирование	67
Дублирование среди разработчиков	68
<b>Тема 10. Ортогональность</b>	69
Что такое ортогональность	69
Преимущества ортогональности	71
Проектирование	72
Инструментальные средства и библиотеки	74
Кодирование	75
Тестирование	76
Документация	76
Как уживаться с ортогональностью	77
<b>Тема 11. Обратимость</b>	79
Обратимость	80
Гибкая архитектура	81
<b>Тема 12. Трассирующие пули</b>	83
Код, сверкающий в темноте	84
Трассирующие пули не всегда попадают в цель	87
Трассирующий код в сравнении с прототипированием	87
<b>Тема 13. Прототипы и памятные записки</b>	89
Что подлежит прототипированию	90
Как пользоваться прототипами	91
Прототипирование архитектуры	91
Как <i>не</i> следует пользоваться прототипами	92
<b>Тема 14. Предметно-ориентированные языки</b>	93
Некоторые предметно-ориентированные языки	94
Характеристики предметно-ориентированных языков	96
Компромисс между внутренними и внешними предметно-ориентированными языками	97
Внутренний предметно-ориентированный язык почти даром	98
<b>Тема 15. Оценивание</b>	100
Какой точности оценки достаточно?	100
Откуда берутся оценки	101
Оценивание сроков выполнения проектов	103
Что ответить на просьбу что-нибудь оценить	105
<b>Глава 3. Основные инструментальные средства</b>	107
<b>Тема 16. Сила простого текста</b>	109
Что такое простой текст	109
В чем сила простого текста	110
Наименьший общий знаменатель	112

<b>Тема 17. Игры в скорлупки</b>	113
Ваша собственная оболочка	114
<b>Тема 18. Эффективное редактирование</b>	116
Что означает свободное владение редактором	116
Стремление к свободному владению редактором	117
<b>Тема 19. Контроль версий</b>	119
Все начинается с исходного кода	120
Ветвление	121
Контроль версий как центральный узел проекта	122
<b>Тема 20. Отладка</b>	124
Психология отладки программ	125
Мысленная установка на отладку	125
С чего начинать отладку	126
Стратегии отладки	127
Программист в чужой стране	128
Бинарный поиск	129
Метод резинового утенка	131
Элемент удивления	133
Контрольный список вопросов по отладке	134
<b>Тема 21. Манипулирование текстом</b>	134
<b>Тема 22. Технические дневники</b>	137
<b>Глава 4. Прагматичная паранойя</b>	139
<b>Тема 23. Проектирование по контракту</b>	140
Принцип проектирования по контракту	141
Реализация проектирования по контракту	145
Проектирование по контракту и аварийное завершение	146
Семантические инварианты	146
Динамические контракты и агенты	148
<b>Тема 24. Мертвые программы не лгут</b>	149
Принцип “поймал–отпустил” — только для ловли рыбы	150
Аварийное завершение вместо отправки на свалку	151
<b>Тема 25. Утвердительное программирование</b>	152
Утверждения и побочные эффекты	153
Оставляйте включенным режим утверждений	154
<b>Тема 26. Как сбалансировать ресурсы</b>	156
Вложенное выделение ресурсов	159
Объекты и исключения	159
Баланс исключений	160
Когда нельзя сбалансировать ресурсы	161
Проверка баланса	162

<b>Тема 27. Не опережайте свет фар вашего автомобиля</b>	163
Черные лебеди	165
<b>Глава 5. Гибкость или ломкость</b>	167
<b>Тема 28. Развязывание</b>	168
Крушения поездов	170
Изъяны глобализации	173
Наследование усугубляет связывание	175
Все дело в изменениях	175
<b>Тема 29. Манипулирование реальным миром</b>	176
События	176
Конечные автоматы	177
Проектный шаблон “Обозреватель”	181
Модель “издатель–подписчик”	182
Реактивное программирование, потоки данных и события	183
События вездесущи	185
<b>Тема 30. Преобразовательное программирование</b>	186
Обнаружение преобразований	189
В чем же здесь польза	193
А как насчет обработки ошибок	194
Преобразования преобразуют программирование	198
<b>Тема 31. Налог на наследование</b>	199
Немного предыстории	199
Трудности применения наследования для совместного использования кода	200
Лучшие альтернативы	202
Наследование редко является ответом	207
<b>Тема 32. Конфигурирование</b>	208
Статическая конфигурация	208
Конфигурация как служба	209
Не пишите морально устаревший код	210
<b>Глава 6. Параллельность</b>	213
<b>Тема 33. Разрывание временного связывания</b>	214
В поисках параллельности	215
Возможности для достижения параллельности	216
Возможности для достижения параллелизма	217
Выявить возможности проще всего	219
<b>Тема 34. Общее состояние — неверное состояние</b>	219
Неатомарные обновления	220
Множественные транзакции ресурсов	224
Обновления без транзакций	225

Другие виды исключительного доступа	226
Доктор, мне больно...	226
<b>Тема 35. Актеры и процессы</b>	227
Актеры могут быть только параллельными	227
Простой актер	228
Отсутствие явной параллельности	232
Erlang подготавливает почву	232
<b>Тема 36. Классные доски</b>	233
Классная доска в действии	235
Системы обмена сообщениями могут быть подобны классным доскам	236
Но не все так просто...	237
<b>Глава 7. По ходу кодирования</b>	239
<b>Тема 37. Прислушайтесь к своим инстинктам</b>	241
Боязнь пустой страницы	241
Борьба с собой	242
Как прислушиваться к своим инстинктам	243
Время играть!	243
Не только <i>свой</i> код	244
Не только код	245
<b>Тема 38. Программирование по совпадению</b>	245
Как программировать по совпадению	246
Как программировать обдуманно	250
<b>Тема 39. Быстродействие алгоритмов</b>	252
Что подразумевается под оценкой алгоритмов	252
Асимптотическое обозначение	253
Разумное оценивание алгоритмов	255
Быстродействие алгоритма на практике	256
<b>Тема 40. Рефакторинг</b>	259
Когда следует выполнять рефакторинг	261
Порядок выполнения рефакторинга	263
<b>Тема 41. Тестировать, чтобы кодировать</b>	265
Обдумывание тестов	265
Кодирование на основе тестов	266
Применяя разработку, на основе тестирования, нужно знать, куда идти	268
Возврат к коду	270
Модульное тестирование	271
Тестирование соответствия контракту	271
Специальное тестирование	273
Создание тестового окна	273
Культура тестирования	274

<b>Тема 42. Тестирование на основе свойств</b>	276
Контракты, инварианты и свойства	276
Генерация тестовых данных	277
Выявление неудачных допущений	278
Тесты на основе свойств способны удивлять	281
Тесты на основе свойств помогают проектировать	282
<b>Тема 43. Будьте осторожны</b>	283
Другие 90%	283
Основные принципы защиты	284
Здравый смысл и криптография	288
<b>Тема 44. Именование</b>	291
Уважение к культуре	293
Согласованность	294
Переименовывать еще труднее	295
<b>Глава 8. До начала проекта</b>	297
<b>Тема 45. Западня требований</b>	298
Миф о требованиях	298
Программирование как терапия	299
Требования — это процесс	300
Поставьте себя на место клиента	301
Требования и правила	302
Требования и реальность	303
Документирование требований	303
Излишне подробная спецификация	305
“Еще одну мятную вафельную пластинку...”	305
Ведение словаря терминов проекта	305
<b>Тема 46. Решение неразрешимых головоломок</b>	307
Степени свободы	308
Идите своим путем!	309
Судьба благоволит подготовленному уму	310
<b>Тема 47. Совместная работа</b>	311
Парное программирование	312
Групповое программирование	313
Что следует делать?	313
<b>Тема 48. Сущность гибкости</b>	315
Гибкий процесс вообще невозможен	316
Что же тогда делать?	317
И это движет проект	318

<b>Глава 9. Прагматичные проекты</b>	319
<b>Тема 49. Прагматичные команды</b>	320
Никаких разбитых окон	321
Сваренные лягушки	321
Планирование пополнения багажа знаний	322
Внешнее общение команды	323
Не повторяйтесь	323
Трассирующие пули в команде	324
Автоматизация	325
Знайте, когда остановиться	325
<b>Тема 50. Кокосами не обойтись</b>	326
Все дело в контексте	327
Один и тот же подход годится не всем	328
Главная цель	329
<b>Тема 51. Начальный набор инструментальных средств программиста-прагматика</b>	331
Ведение проекта путем контроля версий	332
Строгое и непрерывное тестирование	332
Затягивание сетки	336
Полная автоматизация	337
<b>Тема 52. Доставляйте удовольствие своим пользователям</b>	338
<b>Тема 53. Гордость и предубеждение</b>	340
<b>Приложение А. Послесловие</b>	343
Нравственный ориентир	344
Представляйте будущее таким, каким вы хотите его видеть	345
<b>Приложение Б. Библиография</b>	347
<b>Приложение В. Возможные ответы на упражнения</b>	349
<b>Предметный указатель</b>	364

## ПРАГМАТИЧНЫЙ ПОДХОД

---

Имеются определенные рекомендации и приемы, применяемые на всех уровнях разработки программного обеспечения, совершенно универсальные процессы и почти аксиоматические идеи. Но такие подходы редко документируются, а вместо этого они зачастую обнаруживаются в виде отрывочных записей дискуссий о проектировании, управлении проектом или программировании. Но ради вашего удобства мы постараемся здесь собрать все эти идеи и процессы вместе.

Первая и, может быть, самая важная тема касается самой сути разработки программного обеспечения и раскрывается в разделе “Сущность качественного проектирования”. Из нее следует все остальное. Далее следуют разделы “DRY — пороки дублирования” и “Ортогональность”, в которых раскрываются две тесно связанные темы. В первом из них содержится предупреждение не дублировать знания по системам, а во втором — не разделять никакие фрагменты знаний по многим компонентам системы.

По мере увеличения темпов изменений становится все труднее и труднее сохранять приложения в должном состоянии. Поэтому в разделе “Обратимость” мы рассмотрим некоторые методики, помогающие ограждать проекты от их изменяющейся среды.

Два последующих раздела также взаимосвязаны. Так, в разделе “Трассирующие пули” речь пойдет о стиле разработки, позволяющем одновременно собирать требования, проверять проектные решения и реализовывать код. И это единственный способ идти в ногу с современной жизнью. А в разделе “Прототипы и памятные записки” будет показано, каким образом прототипирование применяется для проверки архитектур, алгоритмов, интерфейсов и идей. В современном мире крайне важно проверять идеи и получать ответную реакцию, прежде чем безоговорочно принимать их.

По мере созревания вычислительной техники разработчики во все большем количестве создают языки высокого уровня. И хотя еще не изобретен компилятор, способный принимать команду “сделай это вот так”, в разделе “Предметно-ориентированные языки” представлены более скромные рекомендации, которые вы можете реализовать самостоятельно.

Наконец, все мы работаем в условиях ограниченного времени и ресурсов. Нехватку таких ресурсов можно перенести легче (и принести большее удовлетворение начальству или клиентам), если постараться выяснить, как долго они будут затребованы. Именно об этом и пойдет речь в разделе “Оценивание”.

Упомянутые выше принципы следует непременно иметь в виду во время разработки, чтобы писать более совершенный, быстродействующий и устойчивый код. Вы можете даже сделать его более удобочитаемым.

**ТЕМА 8****СУЩНОСТЬ КАЧЕСТВЕННОГО ПРОЕКТИРОВАНИЯ**

В мире полно умников и знатоков проектирования программного обеспечения, жаждущих передать свою мудрость, приобретенную тяжкими трудами. Для этого существуют особые сокращения, списки (почему-то обычно из пяти пунктов), шаблоны, диаграммы, видеоматериалы, беседы, а возможно, и увлекательные сериалы (Интернет есть Интернет), поясняющие закон Деметры с помощью интерпретирующего танца.

И мы, авторы этой книги, грешим этим. Но нам хотелось бы внести поправки в пояснение того, что лишь стало для нас очевидным. Прежде всего сделаем заявление общего характера.

**Совет 14**

Удачное проектное решение легче изменить, чем неудачное

Вещь считается удачно спроектированной, если она приспособляется к тем, кто ею пользуется. Для кода это означает, что он должен приспособляться к изменениям. Поэтому мы верим в принцип *легкости изменения* (Easier to Change — ETC). Вот и все.

Насколько мы можем судить, всякий принцип проектирования является частным случаем принципа ETC. Чем, например, хорошо уменьшение степени связывания? Оно хорошо тем, что разделяет обязанности, чтобы их легче было изменить. А это и есть принцип ETC. В чем польза принципа единственной ответственности? Она состоит в том, что изменения в требованиях зеркально отображаются в изменениях лишь в одном модуле. И это соответствует принципу ETC. Почему так важно именование? А потому, что удачные имена делают исходный код более удобочитаемым, а ведь его приходится читать, чтобы внести в него изменения. И в этом случае соблюдается принцип ETC!

**Принцип ETC — это ценность, а не правило**

Ценности — это сущности, помогающие принять решение сделать то или другое. Что касается осмысления программного обеспечения, то ETC является

руководящим принципом, помогающим выбрать правильный путь. Как и все остальные ценности, этот принцип должен плавно следовать за здоровой мыслью, мягко подталкивая в правильном направлении.

Но как этого добиться? Как показывает наш опыт, для этого требуется первоначальное сознательное подкрепление. Вам, возможно, придется потратить около недели, сознательно задавая себе вопрос: “Облегчило или же затруднило изменение всей системы в целом то, что я только что сделал?” Делайте это, когда сохраняете файл, пишете тест или исправляете ошибку в программе.

В принципе ЕТС присутствует неявное предположение. Он предполагает, что человек способен выбрать из многих путей именно тот, который будет легче сменить впоследствии. Зачастую здравый смысл позволяет выбрать правильный путь и дает возможность сделать обоснованное предположение. Но иногда на это нет и намека. Что ж, бывает и так. Мы считаем, что в подобных случаях можно сделать две вещи.

Во-первых, если вы не знаете, какую конечную форму примут вносимые вами изменения, то всегда можете вернуться к пути “легкости изменений”: попытаться сделать заменяемым фрагмент кода, который вы пишете. В таком случае, что бы ни произошло впоследствии, этот фрагмент кода не станет непреодолимой преградой на вашем пути. Такая мера кажется крайней, но на самом деле вы должны так или иначе прибегать к ней постоянно. Это всего лишь забота о сохранении кода не связанным и согласованным.

Во-вторых, рассматривайте это как способ развития инстинктов. Запишите данную ситуацию в своем журнале технического учета, упомянув выбранные вами варианты и некоторые предположения об изменениях. Оставьте метку в исходном коде. В дальнейшем, когда этот фрагмент кода придется изменить, вы сможете оглянуться назад и предоставить отчет самому себе. Это может помочь, когда вы в очередной раз окажетесь на аналогичной развилке.

В остальных разделах этой главы рассматриваются конкретные идеи по поводу проектирования. Но все они вызваны описанным здесь одним и тем же принципом легкости изменения.

### ***Другие разделы, связанные с данной темой***

- **Тема 9.** DRY — пороки дублирования.
- **Тема 10.** Ортогональность.
- **Тема 11.** Обратимость.
- **Тема 14.** Предметно-ориентированные языки.
- **Тема 28.** Развязывание, **глава 5** “Гибкость или ломкость”.
- **Тема 30.** Преобразовательное программирование, **глава 5** “Гибкость или ломкость”.
- **Тема 31.** Налог на наследование, **глава 5** “Гибкость или ломкость”.

## Задачи

- Обдумайте принцип проектирования, которым пользуетесь регулярно. Направлен ли он на то, чтобы сделать что-то легко изменяемым?
- Подумайте также о языках и парадигмах программирования: объектно-ориентированного, функционального и т.п. Какие главные положительные, отрицательные или те и другие стороны они имеют для написания кода в соответствии с принципом ЕТС?
- Что вы можете сделать при программировании для того, чтобы исключить отрицательные и выделить положительные стороны<sup>1</sup>?
- Во многих текстовых редакторах имеется (встроенная или через расширения) поддержка команд, выполняемых при сохранении файла. Настройте текстовый редактор на вывод сообщения "ЕТС?" всякий раз<sup>2</sup>, когда сохраняете изменения в файле. Пользуйтесь такой возможностью в качестве напоминания о необходимости обдумать написанный вами код. Насколько легко его изменить?

## ТЕМА 9

## DRY — пороки дублирования

Капитан Джеймс Тиберий Кирк<sup>3</sup> предпочитал снабжать компьютер двумя противоречащими друг другу фрагментами знаний, чтобы нейтрализовать враждебный искусственный интеллект. К сожалению, тот же самый принцип может очень легко погубить *ваш* код.

Программисты обычно собирают, организуют, хранят и используют знания. Они документируют знания в спецификациях, возрождают их в выполняемом коде и пользуются ими для проверок во время тестирования. К сожалению, знания непостоянны. Они изменяются, и зачастую очень быстро. Так, ваше представление о каком-нибудь требовании может измениться после совещания с клиентом. Стоит правительству изменить какой-нибудь нормативный акт, и соответствующая бизнес-логика устаревает. Тесты могут показать, что выбранный алгоритм неработоспособен. Все это непостоянство означает, что нам приходится большую часть времени работать в режиме сопровождения, реорганизации и преобразования знаний в своих системах.

<sup>1</sup> Перефразируя старую песню *Accentuate The Positive* (Делайте акцент на положительном) Джонни Мерсера (Johnny Mercer) — американского исполнителя, популярного в 1940-х годах.

<sup>2</sup> Или хотя бы каждый десятый раз, чтобы оставаться в здравом уме...

<sup>3</sup> Персонаж научно-фантастического телевизионного сериала *Звёздный путь: Оригинальный сериал* (Star Trek: The Original Series). — *Примеч. пер.*

Многие полагают, что сопровождение приложения начинается после его выпуска и означает устранение программных ошибок и совершенствование функциональных средств. Мы считаем, что они не правы. Программисты постоянно работают в режиме сопровождения, поскольку их представления изменяются день за днем, когда поступают новые требования, а уже имеющиеся требования развиваются по мере сосредоточения усилий на проекте. Может измениться и сама среда. Но, независимо от конкретной причины, сопровождение является не каким-то отдельным видом деятельности, а обыкновенной стадией всего процесса разработки.

Когда выполняется сопровождение, приходится искать и изменять представления вещей, т.е. те крупницы знаний, которые вложены в приложение. Но дело в том, что знания очень легко продублировать в спецификациях, процессах и разрабатываемых программах. И делая это, разработчики навлекают сущий кошмар сопровождения, который начинается еще до выпуска приложения. Мы считаем, что единственный способ надежно разрабатывать программное обеспечение и упростить понимание и сопровождение разработок — следовать принципу “не повторяться” (DRY):

*У каждого фрагмента знаний должно быть единственное, недвусмысленное, непререкаемое представление в системе.*

А почему этот принцип называется DRY? А вот почему:

### Совет 15

DRY — Don't Repeat Yourself, т.е. “не повторяйся”

Альтернатива состоит в присутствии чего-то одного в двух или более местах. Если в таком случае изменить что-то одно, то нужно не забыть изменить и все остальное, а иначе программа будет загнана противоречием в угол, как компьютеры пришельцев. И вопрос не в том, чтобы вспомнить об изменениях, а в том, когда они забудутся.

Принцип DRY будет еще не раз упоминаться на страницах этой книги, и зачастую в таком контексте, который не имеет ничего общего с программированием. Мы считаем, что это одно из самых важных инструментальных средств в арсенале программиста-прагматика. И в этом разделе мы вкратце изложим трудности, возникающие в связи с дублированием, а также предложим общие стратегии их преодоления.

## Принцип DRY не только для кодирования

Прежде всего устраним некоторое недоразумение. В первом издании этой книги мы поступили неверно, объяснив лишь, что мы подразумевали под выражением “не повторяться”. И многие посчитали, что принцип DRY имеет отношение только к коду, думая, что он означает “не выполнять копирование и вставку

строк исходного кода”. Но это лишь *часть* принципа DRY, причем мелкая и самая простая. Принцип DRY имеет отношение к дублированию *знаний*, а также *намерений*, т.е. к выражению одного и того же в двух разных местах, кроме того, возможно, разными способами.

В качестве пробного камня попробуйте ответить на следующие вопросы: если должно быть изменено какое-нибудь свойство кода, то не придется ли вносить изменения во многих местах и в самых разных форматах? Не придется ли изменять исходный код и документацию, схему базы данных и хранящуюся в ней структуру и т.д.? Если это именно так, то такой код не соответствует принципу DRY. Рассмотрим некоторые типичные примеры дублирования.

## ДУБЛИРОВАНИЕ В ИСХОДНОМ КОДЕ

Каким бы банальным это ни показалось, но дублирование кода — довольно частое явление. Ниже приведен характерный тому пример.

```
def print_balance(account)
  printf "Debits: %10.2f\n", account.debits
  printf "Credits: %10.2f\n", account.credits
  if account.fees < 0
    printf "Fees: %10.2f-\n", -account.fees
  else
    printf "Fees: %10.2f\n", account.fees
  end
  printf " ——\n"
  if account.balance < 0
    printf "Balance: %10.2f-\n", -account.balance
  else
    printf "Balance: %10.2f\n", account.balance
  end
end
```

Пренебрежем пока что следствиями типичной для новичков ошибки, работая с денежными суммами в числовом формате с плавающей точкой. Вместо этого попробуем выявить дублирование в приведенном выше фрагменте кода. (Мы обнаружили три таких места, но вы можете найти и больше.)

Что же мы нашли? А вот что.

Прежде всего, в рассматриваемом здесь коде явно проявляется дублирование обработки отрицательных чисел методом копирования и вставки. Этот недостаток можно устранить, введя еще одну функцию, как показано ниже.

```
def format_amount(value)
  result = sprintf("%10.2f", value.abs)
  if value < 0
    result + "-"
  else
    result + " "
  end
end
```