Пять строк кода

Роберт Мартин рекомендует

Кристиан Клаусен
Предисловие Роберта С. Мартина

ПИТЕР®
Санкт-Петербург • Москва • Минск
2023

Краткое содержание

Предисловие 16
Введение 18
Благодарности 23
Об авторе 25
Иллюстрация на обложке 26
От издательства 27

Глава 1. Рефакторинг рефакторинга 28
Глава 2. Суть рефакторинга 43

Часть I
Учимся на рефакторинге компьютерной игры

Глава 3. Разбивка длинных функций 54
Глава 4. Пусть код типа работает 77
Глава 5. Совмещение схожего кода 121
Глава 6. Защита данных 174

Часть II
Применение полученных знаний в реальной жизни

Глава 7. Сотрудничество с компилятором 216
Глава 8. Избегайте комментариев 242
Глава 9. Страсть к удалению кода 249
Глава 10. Никогда не бойтесь добавлять код 278
Глава 11. Соблюдение структуры в коде 296
Глава 12. Избегайте оптимизаций и обобщенности 318
Глава 13. Пусть плохой код выглядит плохо 337
Глава 14. Подведение итогов 354
Приложение. Установка инструментов для части I 364

Оглавление

Предисловие 16

Введение 18
Цель: избранные правила и шаблоны рефакторинга 19
Аудитория и план изложения 20
О преподавании 20
О коде 22
Дополнительный проект 22

Благодарности 23

Об авторе 25

Иллюстрация на обложке 26

От издательства 27

Глава 1. Рефакторинг рефакторинга 28
1.1. Что такое рефакторинг 30
1.2. Навыки: что требует рефакторинга 31
1.2.1. Пример запаха кода 32
1.2.2. Пример правила 32
1.3. Культура: когда проводить рефакторинг 33
1.3.1. Рефакторинг старых унаследованных систем 35
1.3.2. Когда рефакторинг делать не нужно 35
1.4. Инструменты: как проводить рефакторинг (безопасно) 36
1.5. Инструменты, необходимые для начала 37
1.5.1. Язык программирования: TypeScript 37

1.5.2. Редактор: Visual Studio Code. 38
1.5.3. Контроль версий: Git 38
1.6. Общий пример: 2D-головоломка 39
1.6.1. Практика ведет к совершенству: вторая база кода 41
1.7. Примечание по реальным программам 41
Резюме 42

Глава 2. Суть рефакторинга. 43
2.1. Улучшение читаемости и обслуживаемости 43
2.1.1. Улучшение кода 44
2.1.2. Обслуживание кода... без изменения его функциональности 47
2.2. Выработка скорости, гибкости и стабильности 47
2.2.1. Выбирайте композицию вместо наследования 48
2.2.2. Изменение кода путем добавления, а не изменения 49
2.3. Рефакторинг и повседневная работа. 50
2.3.1. Рефакторинг как метод для освоения 51
2.4. Определение «области» в контексте программного обеспечения . . . 52
Резюме 52

Часть I
Учимся на рефакторинге компьютерной игры

Глава 3. Разбивка длинных функций 54
3.1. Определяем первое правило: почему пять строк? 55
3.1.1. Правило «Пять строк». 56
3.2. Шаблон проектирования для разбивки функций. 58
3.2.1. Шаблон рефакторинга «Извлечение метода» 63
3.3. Разбивка функций для уравновешивания абстракций 66
3.3.1. Правило «Вызов или передача». 67
3.3.2. Применение правила 68
3.4. Свойства хорошего имени функции 69
3.5. Разбивка функций, делающих слишком много 72
3.5.1. Правило «if только в начале» 72
3.5.2. Применение правила 73
Резюме 76

Глава 4. Пусть код типа работает 77
4.1. Рефакторинг простой инструкции if 78
4.1.1. Правило «Никогда не использовать if с else» 78
4.1.2. Применение правила 80
4.1.3. Шаблон рефакторинга «Замена кода типа классами» 82
4.1.4. Перемещение кода в классы 86
4.1.5. Шаблон рефакторинга «Перемещение кода в классы» 89
4.1.6. Встраивание избыточного метода 93
4.1.7. Шаблон рефакторинга «Встраивание метода» 94
4.2. Рефакторинг большой инструкции if 97
4.2.1. Устранение обобщенности 100
4.2.2. Шаблон рефакторинга «Специализация метода» 102
4.2.3. Допускается только одна инструкция switch 104
4.2.4. Правило «Никогда не использовать switch» 106
4.2.5. Удаление if 108
4.3. Разбираемся с повторением кода 110
4.3.1. Разве нельзя было использовать вместо интерфейсов абстрактные классы? 112
4.3.2. Правило «Наследовать только от интерфейсов» 113
4.3.3. Зачем все это повторение кода? 114
4.4. Рефакторинг двух сложных выражений if 114
4.5. Удаление мертвого кода 118
4.5.1. Шаблон рефакторинга «Пробное удаление с последующей компиляцией» 119
Резюме 120

Глава 5. Совмещение схожего кода 121
5.1. Объединение схожих классов 122
5.1.1. Шаблон рефакторинга «Объединение схожих классов» 130
5.2. Объединение простых условий 136
5.2.1. Шаблон рефакторинга «Совмещение инструкций if» 138
5.3. Объединение сложных условий 139
5.3.1. Использование правил арифметики для условий 140
5.3.2. Правило «Использовать чистые условия» 141
5.3.3. Применение условной арифметики 144
5.4. Объединение кода среди классов 146
5.4.1. Введение диаграмм классов UML для отражения связи классов 151

5.4.2. Шаблон рефакторинга «Введение паттерна “Стратегия”» . . . 153
5.4.3. Правило «Избегать интерфейсов с единственной реализацией» . . . 161
5.4.4. Шаблон рефакторинга «Извлечение интерфейса из реализации» . . . 162
5.5. Объединение похожих функций . . . 165
5.6. Объединение схожего кода . . . 168
Резюме . . . 173

Глава 6. Защита данных . . . 174
6.1. Инкапсуляция с помощью геттеров . . . 175
6.1.1. Правило «Не использовать геттеры или сеттеры» . . . 175
6.1.2. Применение правила . . . 178
6.1.3. Шаблон рефакторинга «Удаление геттера или сеттера» . . . 181
6.1.4. Удаление последнего геттера . . . 183
6.2. Инкапсулирование простых данных . . . 187
6.2.1. Правило «Всегда избегать общих аффиксов» . . . 187
6.2.2. Применение правила . . . 189
6.2.3. Паттерн рефакторинга «Инкапсуляция данных» . . . 194
6.3. Инкапсулирование сложных данных . . . 197
6.4. Устранение инварианта последовательности . . . 204
6.4.1. Шаблон рефакторинга «Обеспечение последовательности» . . . 205
6.5. Устранение перечислений иным способом . . . 208
6.5.1. Перечисление с помощью закрытых конструкторов . . . 208
6.5.2. Переотображение чисел в классы . . . 211
Резюме . . . 213

Часть II
Применение полученных знаний в реальной жизни

Глава 7. Сотрудничество с компилятором . . . 216
7.1. Близкое знакомство с компилятором . . . 217
7.1.1. Слабость: проблема останова имеет ограниченную информативность во время компиляции . . . 218
7.1.2. Сильная сторона: достижимость гарантирует возвращение из методов . . . 219
7.1.3. Сильная сторона: явное присваивание предотвращает обращение к неинициализированным переменным . . . 220

7.1.4. Сильная сторона: контроль доступа помогает инкапсулировать данные 221
7.1.5. Сильная сторона: проверка типов подтверждает свойства . . . 221
7.1.6. Слабость: разыменовывание null рушит приложение 223
7.1.7. Слабость: арифметические ошибки вызывают переполнение или сбои 223
7.1.8. Слабость: ошибки выхода за допустимый диапазон вызывают сбой приложения 224
7.1.9. Слабость: бесконечные циклы стопорят приложение 224
7.1.10. Слабость: взаимные блокировки и состояния гонки вызывают нежелательное поведение 225
7.2. Использование компилятора 227
7.2.1. Подключаем компилятор к работе 228
7.2.2. Не перечьте компилятору 230
7.3. Доверие к компилятору 236
7.3.1. Учим компилятор инвариантам 236
7.3.2. Обращайте внимание на предупреждения 239
7.4. Исключительное доверие к компилятору 240
Резюме 240

Глава 8. Избегайте комментариев 242
8.1. Удаление устаревших комментариев 244
8.2. Удаление закомментированного кода 245
8.3. Удаление бессмысленных комментариев 246
8.4. Преобразование комментариев в имена методов 246
8.4.1. Использование комментариев для планирования 247
8.5. Сохранение комментариев к инвариантам 247
8.5.1. Инварианты в процессе 248
Резюме 248

Глава 9. Страсть к удалению кода 249
9.1. Удаление кода может стать очередной вехой 251
9.2. Удаление кода для устранения ненужной сложности 252
9.2.1. Техническое неведение ввиду неопытности 252
9.2.2. Технические потери из-за нехватки времени 254
9.2.3. Технический долг под давлением обстоятельств 254
9.2.4. Техническая задержка из-за роста 255

9.3. Категоризация кода по степени его близости 256
9.4. Удаление кода в старых унаследованных системах. 257
9.4.1. Прояснение кода с помощью шаблона «Фикус-удавка» 257
9.4.2. Использование «Фикуса-душителя» для улучшения кода . . .260
9.5. Удаление кода из замороженного проекта 261
9.5.1. Получение желаемого результата по умолчанию 261
9.5.2. Минимизация затрат с помощью отрыва и стабилизации . . .262
9.6. Удаление веток в системе контроля версий 262
9.6.1. Минимизация затрат за счет ограничения количества веток. 263
9.7. Удаление документации кода 264
9.7.1. Алгоритм для определения необходимости документирования. 265
9.8. Удаление тестирующего кода 266
9.8.1. Удаление оптимистичных тестов 267
9.8.2. Удаление пессимистичных тестов 267
9.8.3. Исправление или удаление ненадежных тестов. 267
9.8.4. Рефакторинг кода для избавления от плохих тестов. 268
9.8.5. Специализация тестов для их ускорения 268
9.9. Удаление кода дополнительной конфигурации 269
9.9.1. Ограничение конфигурации настраиваемости во времени. 269
9.10. Удаление кода для сокращения числа библиотек 271
9.10.1. Ограничение использования внешних библиотек. 274
9.11. Удаление кода из работающего функционала 275
Резюме 276

Глава 10. Никогда не бойтесь добавлять код 278
10.1. Принятие неуверенности: встретить опасность лицом к лицу 279
10.2. Использование отрыва для преодоления страха создать что-то неправильно 280
10.3. Преодоление страха перед лишними затратами или риском установки фиксированного соотношения 281
10.4. Преодоление страха перед неудачей за счет постепенной разработки 283
10.5. Как копипаст влияет на скорость 284
10.6. Изменение путем добавления через расширяемость 286

10.7. Изменение путем добавления поддерживает обратную совместимость 287
10.8. Изменение путем добавления с помощью переключателей функционала 288
10.9. Изменение путем добавления с помощью ветвления через абстрагирование 292
Резюме 295

Глава 11. Соблюдение структуры в коде 296
11.1. Категоризация структуры на основе области и источника 297
11.2. Три способа, которыми код отражает поведение 298
11.2.1. Выражение поведения в потоке управления 298
11.2.2. Выражение поведения в структуре данных 300
11.2.3. Выражение поведения в данных 303
11.3. Добавление кода для раскрытия структуры 305
11.4. Наблюдение вместо прогнозирования и использование эмпирических техник 306
11.5. Обеспечение безопасности без понимания кода 307
11.5.1. Обеспечение безопасности через тестирование 308
11.5.2. Обеспечение безопасности за счет мастерства 308
11.5.3. Обеспечение безопасности с помощью инструментов 308
11.5.4. Обеспечение безопасности через формальную верификацию 309
11.5.5. Обеспечение безопасности через толерантность к ошибкам . . 309
11.6. Определение неэксплуатируемых структур 309
11.6.1. Эксплуатация пустого пространства с помощью извлечения и инкапсуляции 310
11.6.2. Эксплуатация дублирования с помощью объединения 311
11.6.3. Эксплуатация общих аффиксов с помощью инкапсуляции 314
11.6.4. Эксплуатация типа среды выполнения с помощью динамической диспетчеризации 316
Резюме 317

Глава 12. Избегайте оптимизаций и обобщенности 318
12.1. Стремление к простоте 319
12.2. Когда и как вносить обобщенность 321
12.2.1. Создание минимальной функциональности 322

12.2.2. Объединение компонентов с похожим уровнем стабильности 323
12.2.3. Устранение ненужной обобщенности 323
12.3. Когда и как оптимизировать 323
12.3.1. Рефакторинг перед оптимизацией 324
12.3.2. Оптимизация согласно теории ограничений 326
12.3.3. Координирование оптимизации с помощью метрик 329
12.3.4. Выбор удачных алгоритмов и структур данных 330
12.3.5. Использование кэширования 331
12.3.6. Изоляция оптимизированного кода 333
Резюме 335

Глава 13. Пусть плохой код выглядит плохо 337
13.1. Проблемы привлечения внимания к плохому коду 338
13.2. Разделение на безупречный и legacy-код 339
13.2.1. Теория разбитого окна 340
13.3. Подходы к определению плохого кода 340
13.3.1. Правила в этой книге: простые и конкретные 340
13.3.2. Запахи кода: полноценные и абстрактные 341
13.3.3. Цикломатическая сложность: алгоритмическая (объективная) 342
13.3.4. Когнитивная сложность: алгоритмическая (субъективная) 342
13.4. Правила безопасного ухудшения кода 343
13.5. Методы безопасного ухудшения кода 344
13.5.1. Использование перечислений 344
13.5.2. Использование целых чисел и строк в качестве кода типа 345
13.5.3. Добавление в код магических чисел 346
13.5.4. Добавление в код комментариев 346
13.5.5. Добавление в код пробелов 347
13.5.6. Группировка элементов на основе имен 348
13.5.7. Добавление контекста в имена 349
13.5.8. Создание длинных методов 350
13.5.9. Добавление в методы большого числа параметров 351
13.5.10. Использование геттеров и сеттеров 352
Резюме 353

Глава 14. Подведение итогов 354
14.1. Краткий обзор пройденного 354
14.1.1. Введение: мотивация 355
14.1.2. Часть I: конкретизирование 355
14.1.3. Часть II: расширение горизонтов 355
14.2. Раскрытие внутренней философии 356
14.2.1. Поиск все меньших шагов 356
14.2.2. Поиск внутренней структуры 357
14.2.3. Использование правил для совместной работы 357
14.2.4. Интересы команды важнее личных интересов 358
14.2.5. Простота важнее универсальности 359
14.2.6. Использование объектов или функций высшего порядка . . . 360
14.3. Куда двигаться дальше 361
14.3.1. Микроархитектура 361
14.3.2. Макроархитектура 361
14.3.3. Качество программного обеспечения 361
Резюме 362

Приложение. Установка инструментов для части I 364
Node.js 364
TypeScript 364
Visual Studio Code 365
Git 365
Настройка проекта TypeScript 365
Создание проекта TypeScript 365
Как настроить уровень 366