Содержание

Об авторе 19
О техническом рецензенте 19
Благодарности 20

Введение 21
Ждем ваших отзывов! 22

Часть I. Объекты 23

Глава 1. Проектирование и сопровождение приложений на PHP 25
Проблема 25
PHP и другие языки 28
Об этой книге 32
Объекты 32
Проектные шаблоны 33
Практика 33
Что нового в шестом издании книги 36
Резюме 36

Глава 2. PHP и объекты 39
Неожиданный успех объектов в PHP 39
Вначале был PHP/FI 39
Синтаксические удобства в версии PHP 3 40
Версия PHP 4 и незаметная революция 40
Изменения приняты: PHP 5 43
Заполнение пробела: PHP 7 45
PHP 8: продолжение консолидации 46
Дебаты сторонников и противников объектов 46
Резюме 47

Глава 3. Азы объектов 49
Классы и объекты 49
Первый класс 49
Несколько первых объектов 50
Установка свойств в классе 52
Работа с методами 55

Создание метода конструктора 58
Объявление свойств в конструкторе 60
Аргументы по умолчанию и именованные аргументы 61
Аргументы и типы 63
Примитивные типы данных 63
Дополнительные функции проверки типов 68
Объявления типов: объектные типы 69
Объявления типов: примитивные типы 72
Смешанные типы 75
Объединения 77
Типы, принимающие значение `null` 79
Объявление возвращаемого типа 79
Наследование 80
Проблема наследования 81
Использование наследования 87
Управление доступом к классам: модификаторы `public`, `private` и `protected` 94
Типизированные свойства 98
Резюме 103

Глава 4. Расширенные возможности 105

Статические методы и свойства 105
Константные свойства 111
Абстрактные классы 112
Интерфейсы 115
Трейты 118
Проблема, которую позволяют решить трейты 119
Определение и применение трейтов 120
Применение нескольких трейтов 121
Сочетание трейтов с интерфейсами 122
Устранение конфликтов имен методов с помощью ключевого слова `insteadof` 123
Назначение псевдонимов переопределенным методам трейта 125
Применение статических методов в трейтах 126
Доступ к свойствам класса-хоста 127
Определение абстрактных методов в трейтах 128
Изменение прав доступа к методам трейта 129
Позднее статическое связывание: ключевое слово `static` 130
Обработка ошибок 135
Исключения 137

Завершенные классы и методы 149
Внутренний класс **`Error`** 151
Работа с методами-перехватчиками 152
Определение методов-деструкторов 161
Копирование объектов с помощью метода **`__clone()`** 163
Определение строковых значений для объектов 167
Функции обратного вызова, анонимные функции и замыкания 169
Анонимные классы 177
Резюме 179

Глава 5. Средства для работы с объектами 181

PHP и пакеты 181
Пакеты и пространства имен в PHP 182
Автозагрузка 195
Функции для исследования классов и объектов 201
Поиск классов 203
Получение сведений об объекте или классе 204
Получение полностью квалифицированной строковой ссылки на класс 206
Получение информации о методах 207
Получение информации о свойствах 210
Получение сведений о наследовании 210
Вызов методов 211
Reflection API 214
Краткое введение в Reflection API 214
Время засучить рукава 216
Исследование класса 218
Исследование методов 221
Исследование аргументов методов 223
Использование интерфейса Reflection API 226
Атрибуты 232
Резюме 237

Глава 6. Объекты и проектирование 239

Определение программного проекта 239
Объектно-ориентированное и процедурное программирование 240
Ответственность 246
Связность 247
Тесная связь 247
Ортогональность 248

Выбор классов 248
Полиморфизм 250
Инкапсуляция 253
Забудьте, как это делается 254
Четыре явных признака недоброкачественного кода 256
Дублирование кода 256
Класс, который слишком много знал 256
На все руки мастер 257
Условные инструкции 257
Язык UML 258
Диаграммы классов 258
Диаграмма последовательностей 267
Резюме 270

Часть II. Проектные шаблоны 271

Глава 7. Назначение и применение проектных шаблонов 273

Что такое проектные шаблоны 274
Краткий обзор проектных шаблонов 277
Название 277
Постановка задачи 278
Решение 278
Следствия 279
Формат “Банды четырех” 279
Причины для применения проектных шаблонов 281
Шаблоны определяют задачи 281
Шаблоны определяют решения 281
Шаблоны не зависят от языка программирования 281
Шаблоны определяют словарь 282
Шаблоны проверяются и тестируются 283
Шаблоны предназначены для совместной работы 283
Шаблоны способствуют удачным проектам 284
Шаблоны применяются в распространенных каркасах 284
PHP и проектные шаблоны 284
Резюме 285

Глава 8. Некоторые принципы проектных шаблонов 287

Открытие шаблонов 287
Композиция и наследование 288
Проблема 289
Применение композиции 292

Развязка 296
Проблема 296
Ослабление связанности 298
Программируйте на основе интерфейса, а не его реализации 301
Меняющаяся концепция 303
Проблемы применения шаблонов 304
Шаблоны 305
Шаблоны для формирования объектов 305
Шаблоны для организации объектов и классов 305
Шаблоны, ориентированные на задачи 305
Промышленные шаблоны 305
Шаблоны баз данных 306
Резюме 306

Глава 9. Генерация объектов 307

Формирование объектов: задачи и решения 307
Шаблон Singleton 313
Проблема 314
Реализация 315
Следствия 318
Шаблон Factory Method 319
Проблема 319
Реализация 323
Следствия 326
Шаблон Abstract Factory 326
Проблема 326
Реализация 328
Следствия 331
Шаблон Prototype 333
Проблема 333
Реализация 335
Доведение до крайности: шаблон Service Locator 339
Блестящее одиночество: шаблон Dependency Injection 342
Проблема 342
Реализация 343
Dependency Injection и атрибуты 350
Следствия 360
Резюме 361

Глава 10. Шаблоны для программирования гибких объектов 363
Структурирование классов для повышения гибкости объектов 363
Шаблон Composite 364
Проблема 365
Реализация 368
Следствия 374
Резюме 379
Шаблон Decorator 380
Проблема 380
Реализация 383
Следствия 388
Шаблон Facade 389
Проблема 389
Реализация 392
Следствия 393
Резюме 394

Глава 11. Выполнение задач и представление результатов 395
Шаблон Interpreter 395
Проблема 396
Реализация 397
Трудности реализации шаблона Interpreter 408
Шаблон Strategy 409
Проблема 410
Реализация 411
Шаблон Observer 415
Реализация 418
Шаблон Visitor 426
Проблема 427
Реализация 429
Трудности реализации шаблона Visitor 435
Шаблон Command 436
Проблема 436
Реализация 437
Шаблон Null Object 443
Проблема 444
Реализация 447
Резюме 449

Глава 12. Шаблоны корпоративных приложений 451

Краткий обзор архитектуры 452
Шаблоны 452
Приложения и уровни 453
Нарушение правил с самого начала 457
Шаблон Registry 457
Реализация 460
Уровень представления данных 465
Шаблон Front Controller 466
Шаблон Application Controller 481
Шаблон Page Controller 499
Шаблоны Template View и View Helper 507
Уровень логики приложения 512
Шаблон Transaction Script 512
Шаблон Domain Model 518
Резюме 523

Глава 13. Шаблоны баз данных 525

Уровень хранения данных 525
Шаблон Data Mapper 526
Проблема 527
Реализация 527
Следствия 546
Шаблон Identity Map 548
Проблема 548
Реализация 550
Следствия 553
Шаблон Unit of Work 554
Проблема 554
Реализация 555
Следствия 560
Шаблон Lazy Load 561
Проблема 561
Реализация 562
Следствия 564
Шаблон Domain Object Factory 564
Проблема 565
Реализация 565
Следствия 567

Шаблон Identity Object 569
Проблема 570
Реализация 570
Следствия 577
Шаблоны Selection Factory и Update Factory 578
Проблема 578
Реализация 579
Следствия 584
Что теперь осталось от шаблона Data Mapper 585
Резюме 588

Часть III. Практика 591

Глава 14. Практика — хорошая (и плохая) 593

Не кодом единым 594
Снова изобретаем колесо 594
Ведите себя хорошо 598
Дайте коду крылья 599
Стандарты 601
Vagrant 602
Тестирование 603
Непрерывная интеграция 604
Резюме 605

Глава 15. Стандарты PHP 607

Зачем нужны стандарты 607
Рекомендованные стандарты PHP 609
Особенности рекомендованных стандартов PSR 610
На кого рассчитаны рекомендации стандартов PSR 611
Программирование в избранном стиле 612
Основные рекомендации стандарта PSR-1 по стилю программирования 613
Рекомендации стандарта PSR-12 по стилю программирования 616
Проверка и исправление исходного кода 623
Рекомендации стандарта PSR-4 по автозагрузке 626
Самые важные правила 627
Резюме 630

Глава 16. Создание и использование компонентов PHP средствами Composer 631
Что такое Composer 632
Установка Composer 632
Установка пакетов 633
Установка пакетов из командной строки 634
Версии пакетов 635
Поле **`require-dev`** 637
Composer и автозагрузка 638
Создание собственного пакета 639
Добавление сведений о пакете 640
Пакеты для конкретной платформы 641
Распространение пакетов через сайт Packagist 642
Работа с закрытыми пакетами 645
Резюме 647

Глава 17. Контроль версий средствами Git 649
Зачем нужен контроль версий 650
Установка Git 652
Использование онлайн-хранилища Git 652
Конфигурирование сервера Git 655
Создание удаленного хранилища 656
Подготовка хранилища для локальных пользователей 656
Предоставление доступа пользователям 657
Закрытие доступа к системной оболочке для пользователя **`git`** 658
Начало проекта 659
Клонирование хранилища 663
Обновление и фиксация изменений 664
Добавление и удаление файлов и каталогов 668
Добавление файла 669
Удаление файла 669
Добавление каталога 670
Удаление каталогов 671
Метка о выпуске 671
Ветвление проекта 672
Резюме 681

Глава 18. Тестирование средствами PHPUnit 683
Функциональные и модульные тесты 684
Тестирование вручную 685

Общее представление о PHPUnit 688
Создание контрольного примера 689
Методы утверждений 692
Тестирование исключений 693
Выполнение наборов тестов 695
Ограничения 696
Имитации и заглушки 699
Тесты достигают своей цели, когда завершаются неудачно 704
Написание веб-тестов 708
Рефакторинг кода веб-приложения для тестирования 709
Простое веб-тестирование 712
Введение в Selenium 715
Предупреждения относительно тестирования 723
Резюме 726

Глава 19. Автоматическое построение средствами Phing 727

Назначение Phing 728
Получение и установка Phing 729
Создание документа построения 730
Целевые задания 732
Свойства 735
Типы 745
Задания 752
Резюме 758

Глава 20. Виртуальная машина Vagrant 759

Задача 759
Простейшая установка 761
Выбор и установка образа операционной системы в Vagrant 761
Монтирование локальных каталогов на виртуальной машине Vagrant 764
Подготовка 766
Настройка веб-сервера 768
Настройка сервера баз данных MariaDB 769
Настройка имени хоста 771
Краткие итоги 773
Резюме 774

Глава 21. Непрерывная интеграция 775

Что такое непрерывная интеграция 775
Подготовка проекта к непрерывной интеграции 779

Установка сервера Jenkins 791
Установка модулей, подключаемых к серверу Jenkins 793
Установка открытого ключа доступа к Git 794
Установка проекта 796
Первое построение проекта 800
Настройка отчетов 801
Автоматический запуск процессов построения проектов 805
Резюме 807

Глава 22. Объекты, проектные шаблоны и практика 809

Объекты 809
Выбор 810
Инкапсуляция и делегирование 811
Развязка 811
Повторное использование кода 812
Эстетика 813
Проектные шаблоны 814
Преимущества проектных шаблонов 815
Шаблоны и принципы проектирования 816
Практика 819
Тестирование 820
Стандарты 820
Контроль версий 821
Автоматическое построение 821
Непрерывная интеграция 822
Что упущено из виду 822
Резюме 826

Приложение А. Дополнительные источники информации 827

Литература 827
Статьи 828
Сайты 829

Приложение Б. Простой синтаксический анализатор 833

Сканер 833
Синтаксический анализатор 843

Предметный указатель 861

ГЛАВА 1

Проектирование и сопровождение приложений на PHP

Версия PHP 5.0 была выпущена в июле 2004 года. В ней был внедрен целый ряд радикальных усовершенствований, и самым главным среди них, вероятно, стала коренным образом улучшенная поддержка объектно-ориентированного программирования (ООП). Именно это обстоятельство и вызвало повышенный интерес к объектам и методике ООП в среде разработчиков приложений на PHP. На самом деле это был новый этап развития процесса, начавшегося благодаря выпуску версии 4 языка PHP, в которой ООП впервые стало реальностью.

В этой главе рассмотрен ряд задач, для решения которых требуется методика ООП. В ней вкратце изложена эволюция проектных шаблонов и связанных с ними норм практики программирования. Кроме того, здесь в общих чертах будут обозначены следующие темы, освещаемые в данной книге.

- *Эволюция катастрофы*. Проект не удался.
- *Проектирование и PHP*. Каким образом методика ООП укоренилась в сообществе разработчиков приложений на PHP.
- *Эта книга*. Объекты, шаблоны, практика программирования.

Проблема

Проблема в том, что PHP — очень простой язык. Он искушает вас попробовать реализовать свои идеи и радует хорошими результатами. Большую часть написанного кода на PHP можно встроить непосредственно в разметку веб-страниц, потому что для этого в PHP предусмотрена соответствующая поддержка. Вводя дополнительные функции (например, код доступа к базе данных) в файлы, которые можно перенести с одной страницы на другую, вы не успеете оглянуться, как получите рабочее веб-приложение.

И, тем не менее, вы уже стоите на пути к краху. Конечно, вы этого не осознаете, потому что ваш сайт выглядит потрясающе. Он прекрасно работает, заказчики довольны, а пользователи тратят деньги.

Трудности возникают, когда вы возвращаетесь к исходному коду, чтобы начать новый этап разработки. Теперь ваша команда увеличилась, пользователей стало больше да и бюджет вырос. Но, если не принять меры, все пойдет прахом. Образно говоря, все выглядит так, как будто ваш проект был отравлен.

Ваш новый программист изо всех сил старается понять код, который вам кажется простым и естественным, хотя, возможно, местами слегка запутанным. И этому новому программисту требуется больше времени, чем вы рассчитывали, чтобы войти в курс дела и стать полноправным членом команды. На простое изменение, которое вы рассчитывали сделать за один день, уходит три дня, потому что вы обнаруживаете, что в результате нужно обновить порядка двадцати веб-страниц.

Один из программистов сохраняет свою версию файла поверх тех серьезных изменений, которые вы внесли в тот же код немного раньше. Потеря обнаруживается только через три дня, к тому времени как вы изменили собственную локальную копию файла. Еще один день уходит на то, чтобы разобраться в этом беспорядке, а между тем без дела сидит третий программист, который также работал с этим файлом.

Ваше веб-приложение пользуется популярностью, и поэтому вам нужно перенести его исходный код на новый сервер. Устанавливать на нем свой проект приходится вручную, и в этот момент вы обнаруживаете, что пути к файлам, имена баз данных и пароли жестко закодированы во многих исходных файлах. В результате вы останавливаете работу своего веб-сайта на время переноса исходного кода, потому что не хотите затереть изменения в конфигурации, которых требует такой перенос. Работа, которую планировалось выполнить за два часа, в итоге растягивается на восемь часов, поскольку обнаружилось, что кто-то умный задействовал модуль `ModRewrite` веб-сервера Apache, и теперь для нормальной работы веб-приложения требуется, чтобы этот модуль функционировал на сервере и был правильно настроен.

В конечном счете вы успешно преодолеваете второй этап разработки. И полтора дня все идет хорошо. Первое сообщение об ошибке приходит в тот момент, когда вы собираетесь уходить с работы домой. Еще через минуту звонит заказчик с жалобой. Его сообщение об ошибке напоминает предыдущее, но в результате более тщательного анализа обнаруживается,

что это другая ошибка, которая вызывает схожее поведение. Вы вспоминаете о простом изменении в начале данного этапа, которое потребовало серьезно модифицировать остальную часть проекта.

И тогда вы понимаете, что модифицировано было не все. Это произошло либо потому, что некоторые моменты были упущены в самом начале, либо потому, что изменения в проблемных файлах были затерты в процессе объединения. В страшной спешке вы вносите изменения, необходимые для исправления ошибок. Вы слишком спешите и не тестируете внесенные изменения. Ведь это же простые операции копирования и вставки, что тут может случиться?

Придя на работу на следующее утро, вы выясняете, что модуль корзины для покупок не работал всю ночь. Оказалось, что, внося вчера изменения в последнюю минуту, вы пропустили открывающие кавычки, в результате чего код стал неработоспособным. И пока вы спали, потенциальные клиенты из других часовых поясов бодрствовали и были готовы потратить деньги в вашем интернет-магазине. Вы исправляете ошибку, успокаиваете заказчика и мобилизуете команду на "пожарные" работы в течение еще одного дня.

Подобная история о ежедневных буднях программистов может показаться преувеличением, но все это мне случалось наблюдать неоднократно. Многие проекты на PHP начинались с малого, а затем превращались в настоящих монстров!

В данном проекте логика приложения содержится также на уровне представления данных, и поэтому дублирование происходит уже в коде запросов к базе данных, проверке аутентификации, обработке форм, причем этот код копируется с одной страницы на другую. Всякий раз, когда требуется внести коррективы в один из таких блоков кода, это приходится делать везде, где присутствует такой код, иначе неминуемо возникнет ошибка.

Отсутствие документации затрудняет понимание исходного кода, а недостаточность тестирования оставляет скрытые дефекты необнаруженными вплоть до момента развертывания приложения. Изменение основного направления деятельности заказчика часто означает, что в результате модификации прикладной код меняет свое первоначальное назначение и в конце концов начинает выполнять задачи, для которых он изначально вообще не был предназначен. А поскольку такой код, как правило, разрабатывался и развивался в качестве единой гремучей смеси, в которой было много чего намешано, то очень трудно, или даже невозможно, перестро-

иться и переписать отдельные его фрагменты, чтобы он соответствовал новой цели.

Но все это не так уж и плохо, если вы — независимый консультант по PHP. Анализ и исправление подобных систем позволят вам покупать дорогие напитки и наборы DVD в течение полугода или даже дольше. А если серьезно, то проблемы подобного рода как раз и отличают удачную коммерческую деятельность от неудачной.

PHP и другие языки

Феноменальная популярность языка PHP означает, что он был основательно протестирован во всех сферах своего применения еще на начальном этапе своего развития. Как поясняется в следующей главе, язык PHP начинался как набор макрокоманд для управления персональными веб-страницами. С появлением версии PHP 3, а в значительной степени — PHP 4, этот язык быстро стал популярным и мощным инструментом для создания веб-сайтов крупных коммерческих компаний. Но во многих случаях область его применения ограничивалась разработкой сценариев и средств управления проектами. В некоторых кругах специалистов PHP заслужил несправедливую репутацию языка для любителей, который лучше всего приспособлен для задач представления данных.

Примерно в то же время, когда наступило новое тысячелетие, в других сообществах программистов стали распространяться новые идеи. Интерес к объектно-ориентированному проектированию всколыхнул сообщество программирующих на Java. Вам такой интерес может показаться излишним, поскольку Java и так является объектно-ориентированным языком. Но ведь Java задает лишь рациональное зерно, которым нужно еще научиться правильно пользоваться, поскольку применение в программах классов и объектов само по себе не определяет конкретный подход к проектированию.

Понятие проектного шаблона как способа описания задачи вместе с сутью ее решения впервые обсуждалось еще в 1970-х годах. Пожалуй, можно считать удачей, что первоначально эта идея возникла в области строительства и архитектуры, а не в вычислительной технике, появившись в работе Кристофера Александера (Christopher Alexander) *A Pattern Language* в 1977 году. В начале 1990-х годов приверженцы ООП стали применять аналогичные методики для определения и описания задач

разработки программного обеспечения. Основополагающая книга по проектным шаблонам, *Design Patterns: Elements of Reusable Object-Oriented Software*[1], опубликованная в 1995 году под авторством "Банды четырех", незаменима и по сей день. Шаблоны, которые в ней описаны, обязательны для каждого, кто делает первые шаги в данной области. Именно поэтому большинство шаблонов, описываемых в данной книге, взято из этого фундаментального труда.

В интерфейсах API языка Java изначально применяются многие основные шаблоны, но до конца 1990-х годов проектные шаблоны так и не были полностью осмыслены сообществом программистов. Книги по проектным шаблонам быстро заполонили отделы компьютерной литературы в книжных магазинах, и на форумах программистов появились первые признаки горячей полемики со словами похвалы или неодобрения.

Возможно, вы думаете, что шаблоны — это эффективное средство передачи специальных знаний предметной области или, наоборот, "мыльный пузырь" (какой точки зрения придерживаюсь лично я, видно из названия данной книги). Каким бы ни было ваше мнение, трудно отрицать, что подход к разработке программного обеспечения, который поощряется в шаблонах, полезен сам по себе.

Не менее популярными для обсуждения стали и родственные темы. К их числу относится методика экстремального программирования (Extreme Programming — XP), одним из авторов которой является Кент Бек (Kent Beck)[2]. Экстремальное программирование — это подход к проектированию, который направлен на гибкое, проектно-ориентированное, очень тщательное планирование и выполнение. Один из главных его принципов настаивает на том, что ключевым фактором для успешного выполнения проекта является тестирование. Тесты должны быть автоматическими и выполняться часто; и желательно, чтобы они были разработаны до написания самого кода.

Еще одно требование методики экстремального программирования состоит в том, что проекты должны быть разбиты на небольшие (попросту — мелкие) повторяющиеся стадии. А код и требования к нему должны постоянно анализироваться. Архитектура и проект должны быть предме-

---

[1] Гамма, Э., Хелм, Р., Джонсон, Р., Влиссидес, Д. *Приемы объектно-ориентированного проектирования. Паттерны проектирования* : Пер. с англ. — Изд-во "Питер", 2007.

[2] Бек, К. *Экстремальное программирование* : Пер. с англ. — Изд-во "Питер", 2007.

том постоянного совместного обсуждения, что ведет к частому пересмотру разрабатываемого кода.

Методика экстремального программирования стала воинствующим крылом в движении сторонников проектирования программного обеспечения. Что касается умеренной тенденции, то она представлена в одной из лучших книг по программированию, которые я когда-либо читал: *The Pragmatic Programmer* (Andrew Hunt, David Thomas, издательство Addison-Wesley Professional, 1999)[3].

Некоторые считают, что методика экстремального программирования была в какой-то степени культовым явлением, но за два десятилетия практики объектно-ориентированного программирования она достигла высочайшего уровня, а ее принципы широко использовались и заимствовались. В качестве мощного дополнения к шаблонам был использован процесс реорганизации кода, называемый *рефакторингом*. Рефакторинг развивался с 1980-х годов, но только в 1999 году он был кодифицирован в каталоге шаблонов рефакторинга, который был опубликован Мартином Фаулером (Martin Fowler) в книге *Refactoring: Improving the Design of Existing Code*[4] и определил данную область проектирования программного обеспечения.

С развитием и ростом популярности методики экстремального программирования и проектных шаблонов тестирование также стало злободневным вопросом. Особое значение автоматических тестов еще более усилилось с появлением мощной тестовой платформы JUnit, которая стала главным инструментом в арсенале средств программистов на Java. Основополагающая статья *Test Infected: Programmers Love Writing Tests* (Kent Beck, Erich Gamma; `http://junit.sourceforge.net/doc/testinfected/testing.htm`) стала превосходным введением в этот предмет и до сих пор не потеряла своей актуальности.

Примерно в то же время вышла более эффективная версия PHP 4 с усиленной поддержкой объектов. Благодаря этим усовершенствованиям появилась возможность создавать полностью объектно-ориентированные приложения. Программисты воспользовались этой возможностью, к некоторому удивлению создателей ядра Zend Зеэва Сураски (Zeev Suraski) и Энди Гутманса (Andi Gutmans), которые присоединились к Расмусу Лер-

---

[3] Хант, Э., Томас, Д. *Программист-прагматик: 2-е юбилейное изд.*, : Пер. с англ. — Изд-во "Диалектика", 2020.

[4] Фаулер, М., Бек, К., Брант, Д., Опдайк, У., Робертс, Д. *Рефакторинг: улучшение проекта существующего кода* : Пер. с англ. — Изд-во "Диалектика", 2017.

дорфу (Rasmus Lerdorf) для разработки PHP. Как поясняется в следующей главе, поддержка объектов в PHP оказалась далеко не идеальной, но при строгой дисциплине и аккуратном применении синтаксиса на PHP можно было все же писать объектно-ориентированные программы.

Тем не менее неприятности, подобные описанной в начале этой главы, случались часто. Культура проектирования была еще недостаточно развита, и в книгах по PHP о ней едва ли упоминалось. Но в Интернете интерес к этому предмету был очевиден. Леон Аткинсон (Leon Atkinson) написал статью о PHP и проектных шаблонах для Zend в 2001 году, а Гарри Фойкс (Harry Fuecks) завел в 2002 году журнал по адресу `www.phppatterns.com` (он уже давно прекратил свое существование, хотя сайт по указанному адресу по-прежнему действует). В конечном итоге стали появляться проекты на основе шаблонов (например, BinaryCloud), а также инструментальные средства для автоматического тестирования и документирования разрабатываемых приложений.

Выход первой бета-версии PHP 5 в 2003 году обеспечил будущее PHP в качестве языка для объектно-ориентированного программирования. Процессор Zend Engine 2 обеспечил существенно улучшенную поддержку объектов. И, что не менее важно, его появление стало сигналом, что объекты в частности и методика объектно-ориентированного программирования вообще могут служить основанием для любого проекта на PHP.

С годами версия PHP 5 продолжала развиваться и совершенствоваться. В ней появились такие важные средства, как поддержка пространств имен и механизма замыканий. За это время версия PHP 5 завоевала репутацию надежного и самого лучшего средства для разработки серверных веб-приложений.

Эта тенденция была продолжена в версии PHP 7, выпущенной в декабре 2015 года. В частности, в этой версии поддерживаются объявления типов параметров и возвращаемых типов — возможности, внедрения которых многие годы настоятельно требовали многие разработчики, о чем упоминалось в том числе на страницах предыдущих изданий данной книги. Есть много других возможностей и улучшений, включая анонимные классы, улучшенное использование памяти и повышение скорости. С годами, с точки зрения объектно-ориентированного разработчика, язык постоянно становится все более надежным, ясным и приятным в работе.

В декабре 2020 года, почти через пять лет после выпуска PHP 7, был подготовлен к выпуску PHP 8. Хотя некоторые детали реализации могут измениться (и изменились за время написания этой книги), основные его

функциональные возможности уже доступны и подробно описаны в этой книге. Они включают улучшения объявления типов, оптимизированное назначение свойств и многие другие новые функции.

Об этой книге

Эта книга не является попыткой открыть что-то новое в области объектно-ориентированного программирования, и в этом отношении она просто "стоит на плечах гигантов". Ее цель — исследовать в контексте PHP некоторые устоявшиеся принципы проектирования и основные шаблоны проектирования (особенно те, которые упоминаются в книге *Design Patterns* — классическом труде "Банды четырех"). В конце книги будет предпринята попытка выйти за пределы строгих ограничений прикладного кода, чтобы рассмотреть инструментальные средства и методики, которые могут помочь в работе над проектом. За исключением этого введения и краткого заключения, данная книга разделена на три основные части: "Объекты", "Проектные шаблоны" и "Практика".

Объекты

Часть I начинается с краткой истории развития PHP и объектов — с их превращения из дополнительной возможности в версии PHP 3 в основную начиная с версии PHP 5. Вы можете быть опытным программистом и свободно писать программы на PHP, но при этом очень мало знать или почти ничего не знать об объектах. Именно по этой причине данная книга начинается с основных принципов, — чтобы объяснить, что такое объекты, классы и наследование. Даже на этой начальной стадии в данной части книги рассматриваются некоторые усовершенствования объектов, появившиеся в версиях PHP 5, PHP 7 и PHP 8.

После основ объекты будут рассмотрены более углубленно в ходе исследования более развитых объектно-ориентированных возможностей PHP. Отдельная глава в этой части книги посвящена инструментальным средствам, которые предусмотрены в PHP для работы с объектами и классами.

Но знать, как объявить класс и как его использовать для создания экземпляра объекта, еще недостаточно. Сначала необходимо правильно выбрать участников для разрабатываемой системы и определить оптимальные способы их взаимодействия. Описать и усвоить эти варианты выбора намного

труднее, чем простые факты об инструментальные средствах и синтаксисе объектов. Данная часть завершается введением в объектно-ориентированное проектирование на PHP.

Проектные шаблоны

Шаблон описывает задачу, поставленную в проекте программного обеспечения, предоставляя саму суть решения. "Решение" здесь не является частью кода, которую можно найти в справочнике, вырезать и вставить (хотя на самом деле справочники — превосходные ресурсы для программистов). В проектном шаблоне описывается подход, который можно использовать для решения поставленной задачи. К этому может прилагаться пример реализации, но он менее важен, чем концепция, которую он призван иллюстрировать.

Часть II начинается с определения проектных шаблонов и описания их структуры. В ней рассматриваются также некоторые причины их широкой распространенности.

При создании шаблонов обычно выдвигаются некоторые основополагающие принципы проектирования, которых следует придерживаться в процессе разработки всего приложения. Уяснение этого факта поможет лучше понять причины создания шаблона, который затем можно применять при создании любой программы. Некоторые из этих принципов обсуждаются в данной части книги, где представлен также унифицированный язык моделирования (Unified Modeling Language — UML) — платформенно-независимый способ описания классов и их взаимодействия.

Несмотря на то что данная книга не является справочником по проектным шаблонам, в ней все же рассматриваются некоторые из самых известных и полезных шаблонов. Сначала описывается задача или проблема, для решения которой предназначен каждый шаблон, а затем анализируется ее решение и демонстрируется пример его реализации на PHP.

Практика

Даже самое гармоничное архитектурное сооружение разрушится, если не обращаться с ним должным образом. В части III представлены инструментальные средства, с помощью которых можно создать структуру, способную обеспечить удачное завершение проекта. Если в остальных частях этой книги речь идет о практике проектирования и программирования, то

в этой части — о практике сопровождения прикладного кода. Рассматриваемые здесь инструментальные средства позволяют создавать поддерживающую инфраструктуру проекта, помогая обнаруживать ошибки по мере их проявления, способствуя сотрудничеству программистов и обеспечивая простоту установки и ясность прикладного кода.

Ранее вкратце упоминалось об эффективности автоматических тестов. Данная часть начинается со вступительной главы, в которой представлен краткий обзор насущных задач и решений в данной области разработки программного обеспечения.

Многие программисты поддаются искушению делать все самостоятельно. А диспетчер зависимостей Composer совместно с главным хранилищем пакетов Packagist обеспечивает доступ к тысячам пакетов с управляемыми зависимостями, из которых легко составляются целые проекты. В этой части обсуждаются достоинства и недостатки самостоятельной реализации в сравнении с развертыванием пакетов средствами Composer. Здесь, в частности, описывается применяемый в Composer механизм установки, упрощающий процесс развертывания пакетов вплоть до выполнения единственной команды.

Код — это продукт коллективного труда. С одной стороны, этот факт приносит внутреннее удовлетворение, а с другой — может превратить все в сущий кошмар. Система контроля версий Git предоставляет группе программистов возможность работать совместно над одним и тем же кодом, не рискуя затереть результаты работы друг друга. Она позволяет получать моментальные снимки проекта на любой стадии его разработки, видеть, кто и какие изменения вносил, а также разбивать проект на независимые ветки, которые затем можно объединить. В системе Git сохраняется ежедневное состояние проекта.

Когда разработчики совместно работают над приложениями или библиотеками, они нередко вносят разные условные обозначения и стили оформления кода в общий проект. И хотя в таком явлении нет ничего дурного, оно подрывает основы организации взаимодействия. От понятий *совместимости* и *соответствия* нередко бросает в дрожь, но непреложным является тот факт, что творческая инициатива в Интернете опирается все же на стандарты. Соблюдая определенные соглашения, можно свободно экспериментировать в невообразимо обширной “песочнице”. Поэтому в новой главе этой части исследуются стандарты PHP, польза от них для разработчиков и причины, по которым они должны строго соблюдать *соответствие* стандартам.

Существуют две неизбежные проблемы. Во-первых, ошибки часто повторяются в одном и том же фрагменте кода, из-за чего некоторые рабочие дни проходят с ощущением повторения уже однажды пройденного. И во-вторых, улучшения часто портят столько же или даже больше, чем исправляют. Автоматическое тестирование помогает решить обе эти проблемы, обеспечивая систему раннего предупреждения об ошибках в прикладном коде. В этой части книги представлена PHPUnit — эффективная реализация так называемой тестовой платформы xUnit, первоначально предназначавшейся для Smalltalk, а теперь приспособленной для многих языков, особенно для Java. Здесь рассматриваются функции PHPUnit в частности и преимущества тестирования вообще, а также цена, которую приходится за него платить.

Для приложений характерен беспорядок. Им могут понадобиться файлы, которые необходимо скопировать в нестандартные места, базы данных или изменение конфигурации сервера. Короче говоря, установка приложений требует *немалых* хлопот. Утилита Phing представляет собой точную переносимую версию утилиты Ant, применяемой в проектах на Java. Обе утилиты, Phing и Ant, интерпретируют файл построения и обрабатывают исходные файлы любым заданным вами способом. Чаще всего их нужно скопировать из исходного каталога в различные системные каталоги. Но если ваши потребности возрастут, то утилита Phing сможет легко их удовлетворить.

В некоторых компаниях строго придерживаются вполне определенных платформ разработки, но зачастую команды разработчиков работают в самых разных операционных системах. Например, подрядчики могут быть оснащены портативными персональными компьютерами (как у Пола Трегоинга, технического рецензента пятого и настоящего изданий книги), а некоторые члены команд разработчиков бесконечно пропагандируют (это дело религии) свой излюбленный дистрибутив Linux (как, например, я — версию Fedora). Многие из них мечтают приобрести новый и привлекательный MacBook Air, чтобы произвести впечатление на окружающих во время деловых встреч за чашкой кофе или производственных совещаний (хотя это совсем не делает их последователями моды современной молодежи). И на всех этих платформах может запускаться комплект серверного программного обеспечения LAMP с разной степенью трудности. Но в идеальном случае разработчики должны отлаживать свой код в той среде, которая очень похожа на конечную производственную систему. Поэтому здесь рассматривается инструментальное средство Vagrant, в котором при-

меняется виртуализация для того, чтобы члены команды разработчиков могли работать на своих излюбленных платформах разработки, но выполнять прикладной код общего проекта в системе, похожей на производственную.

Тестировать и создавать проект — это, конечно, хорошо, но для того чтобы извлечь из этого пользу, необходимо установить и непрерывно выполнять наборы тестов. Если не автоматизировать процесс создания и тестирования проекта, все очень быстро пойдет прахом. Поэтому здесь рассматривается ряд средств и методик, которые совместно называются *непрерывной интеграцией* и призваны помочь справиться с подобной задачей.

Что нового в шестом издании книги

PHP — это живой язык, и все его средства постоянно обновляются, изменяются и дополняются. Поэтому новое издание книги было тщательно пересмотрено и основательно обновлено, чтобы описать все новые изменения и возможности PHP.

В настоящем издании описываются новые возможности PHP, такие как атрибуты и многочисленные усовершенствования в области объявления типов. В соответствующих примерах используются средства PHP 8 (там, где это уместно), поэтому имейте в виду, что вам часто придется выполнять исходный код в интерпретаторе PHP 8. В противном случае будьте готовы к тому, чтобы изменить код и сделать его совместимым с предыдущей версией PHP.

Резюме

Эта книга посвящена объектно-ориентированному проектированию и программированию. В ней описаны средства для работы с кодом PHP, начиная с приложений для совместной работы программистов и заканчивая утилитами для развертывания кода.

Эти две темы раскрывают одну и ту же проблему под различными, но дополняющими друг друга углами зрения. Основная цель состоит в том, чтобы создавать системы, отвечающие заданным требованиям и обеспечивающие возможности совместной работы над проектами.

Второстепенная цель состоит в достижении эстетичности систем программного обеспечения. Программисты создают продукты, облекая их в

определенную форму и приводя их в нужное действие. Они тратят немало рабочего времени, воплощая эти формы в жизнь. Программисты создают нужные им средства — отдельные классы и объекты, компоненты программного обеспечения или окончательные продукты, — чтобы соединить их в единое изящное целое. Процесс контроля версий, тестирования, документирования и построения представляет собой нечто большее, чем достижение этой цели. Это часть формы, которой требуется достичь. Необходимо иметь не только ясный и понятный код, но и кодовую базу, предназначенную как для разработчиков, так и для пользователей. А механизм совместного распространения, чтения и развертывания проекта должен иметь такое же значение, как и сам прикладной код.