*Посвящается Синди*
*— Мартин*

# Предисловие

Предметно-ориентированные языки (Domain-Specific Languages — DSL) были частью компьютерного мира еще до того, как я научился программировать. Спросите ветеранов Unix или Lisp, и они будут счастливы утомить вас до судорог рассказами о том, как предметно-ориентированные языки помогали им в их работе и какие трюки с их помощью они ухитрялись выполнять. Тем не менее этим языкам не суждено было стать заметной частью упомянутого компьютерного мира. Большинство людей знают о предметно-ориентированных языках только по чьим-то рассказам, и часто эти рассказы ограничиваются описанием только одной из сторон имеющихся методов.

Я написал эту книгу в надежде изменить ситуацию и предоставить вам как можно больше информации о методах предметно-ориентированных языков, чтобы вы могли принять осознанное решение об их использовании в своей работе и о том, какие именно методы предметно-ориентированных языков применять.

Предметно-ориентированные языки популярны по нескольким причинам, но я остановлюсь только на двух из них: повышение производительности труда разработчиков и улучшение связи с экспертами в предметной области. Правильно выбранный язык может сделать сложный блок кода существенно проще для понимания, что повышает производительность работающих с ним. Он также упрощает общение разработчика программного обеспечения со специалистами в предметной области, по сути предоставляя текст, который одновременно действует и как выполняемое программное обеспечение, и как описание проблемы, которое узкие специалисты в данной области знаний могут прочесть, чтобы понять, как их идеи представлены в программной системе. Трудно переоценить возможность говорить со специалистами на одном языке — эта возможность разрушает самый высокий из барьеров на пути общения между программистами и их клиентами и устраняет массу узких мест в разработке специализированного программного обеспечения.

Однако не хотелось бы и преувеличивать значение предметно-ориентированных языков. Я часто говорю, что всякий раз, рассматривая преимущества применения предметно-ориентированного языка для решения той или иной проблемы, вы должны выполнить еще одно рассмотрение, подставив на этот раз вместо слов “предметно-ориентированный язык” слово “библиотека”. Многое из того, что вы получаете с помощью предметно-ориентированного языка, можно получить, создав соответствующую программную структуру. Большинство предметно-ориентированных языков на самом деле представляют собой просто косметический фасад над библиотеками или программной структурой. В результате выгоды от предметно-ориентированных языков часто оказываются меньшими, чем думают неискушенные в этих вопросах люди; однако в любом

случае для принятия решения нужно четко понимать, какие именно положительные и отрицательные стороны имеет каждое из решений. Знание хорошо себя зарекомендовавших технологий существенно снижает стоимость построения предметно-ориентированного языка (и я надеюсь, что моя книга поможет вам получить эти знания). Фасад может быть декоративной деталью, но часто он так же полезен, как и само здание.

## Почему сейчас ?

Предметно-ориентированные языки известны издавна, но в последние годы они породили значительный всплеск интереса. И именно в это время я решил потратить пару лет на эту книгу. Я не уверен, что смогу пояснить, чем вызван такой интерес к предметно-ориентированным языкам, но я поделюсь своей личной точкой зрения.

На рубеже нового тысячелетия возникло ощущение превосходящей разумные рамки стандартизации в языках программирования, по крайней мере в моем мире корпоративного программного обеспечения. Несколько лет Java рассматривался как Единственный Язык Будущего, и даже когда Microsoft выпустила свой C#, он был очень похож на Java. В новых разработках доминирующую позицию занимали компилируемые статические объектно-ориентированные языки программирования с C-образным синтаксисом (даже Visual Basic не избежал попыток приведения его к этому общему знаменателю).

Но вскоре стало ясно, что не все в порядке в королевстве Java/C#. В наличии имелась логика, не хотевшая укладываться в ложе этих языков программирования, что привело к возникновению конфигурационных файлов в формате XML. Программисты шутили, что в результате они пишут больше строк XML, чем на Java/C#. Отчасти это было связано с желанием изменить поведение программы во время выполнения, а отчасти — с желанием выразить аспекты поведения более дружественным и простым для пользователя способом. XML, несмотря на его "зашумленный" синтаксис, позволяет определять собственный словарь и предоставляет строгую иерархическую структуру.

Но "зашумленность" XML все же оказалась слишком большой. Люди жаловались на то, что от угловых скобок у них рябит в глазах, и возникло желание получить преимущества XML-файлов конфигурации без затрат, связанных с применением XML.

Наш рассказ постепенно достиг взрывоподобного явления Ruby On Rails. Независимо от области применения этот язык оказывал огромное влияние на работу программистов с библиотеками и структурными схемами. Бо́льшая часть методов работы сообщества Ruby представляет собой более гибкий подход, при котором взаимодействие с библиотекой должно было походить на программирование на специализированном языке. Этот способ мышления восходит к одному из старейших языков программирования — Lisp. При этом подходе и способе мышления можно увидеть цветение даже на каменистой почве Java/C#: в обоих языках все более популярными становятся интерфейсы, вероятно, из-за влияния таких продуктов, как JMock и Hamcrest.

Узнав обо всем этом, я почувствовал, что у меня имеется немалый пробел в знаниях. Я видел людей, использующих XML там, где более удобным и ничуть не более трудным было бы применение иного синтаксиса. Я видел людей, пытавшихся загнать Ruby в тесные рамки сложных выражений, в которых гораздо легче было бы применить пользовательский синтаксис. Я видел программистов, играющихся с синтаксическими анализаторами там, где можно было бы обойтись существенно меньшим количеством работы с гибкими интерфейсами в их обычных языках программирования.

Мне кажется, что причина всех этих перегибов — в недостатке знаний. Квалифицированные программисты недостаточно знакомы с технологиями предметно-ориентирован-

ных языков, чтобы принять обоснованное решение о том, какие из них использовать. Этот пробел в знаниях я бы и хотел восполнить.

## Значение предметно-ориентированных языков

Более подробно на эту тему мы поговорим в главе 2, "Использование предметно-ориентированных языков", а пока что я вижу две основные причины, по которым вы должны быть заинтересованы в предметно-ориентированных языках (а значит, и в методах, описанных в этой книге).

Первая причина заключается в повышении производительности труда программиста. Рассмотрим фрагмент кода

```
input =~ /\d{3}-\d{3}-\d{4}/
```

Вы можете распознать в нем регулярное выражение, и, возможно, вы знаете, что оно означает. Регулярные выражения часто критикуют за излишнюю загадочность, но представьте, как бы выглядел обычный код сопоставления шаблону без регулярных выражений. Насколько легко было бы понять и модифицировать такой код в сравнении с регулярным выражением?

Предметно-ориентированные языки позволяют сделать некоторые частные задачи программирования более легкими для понимания, а значит, соответствующие программы можно будет быстрее писать, легче изменять, и они будут менее подвержены ошибкам.

Вторая причина заинтересованности в предметно-ориентированных языках выходит за рамки программирования. Поскольку такие языки меньше и их легче понимать, они позволяют специалистам в предметной области, не являющимся программистами, понимать код, управляющий важными аспектами их профессии. Анализ реального кода со специалистами в предметной области позволит значительно обогатить и сделать более полезным общение между программистами и их клиентами.

Обсуждая подобные темы, люди часто говорят, что предметно-ориентированные языки позволят полностью избавиться от программистов. Я очень скептически отношусь к этому аргументу; в конце концов, то же самое в свое время говорили и о COBOL. Хотя, конечно, есть языки (такие, как CSS), написанные людьми, которые не называют себя программистами. Однако для предметно-ориентированных языков чтение написанного кода имеет большее значение, чем написание. Если эксперты в предметной области читают и в основном понимают написанный программистами код, то им будет гораздо легче общаться с программистом, который этот код создавал.

Вторая причина использования предметно-ориентированных языков не так проста, но награда стоит затраченных усилий. Общение между программистами и их клиентами — самое узкое место в разработке программного обеспечения, так что все, что может облегчить такое общение, стоит затраченных усилий.

## Не бойтесь объема этой книги

Толщина этой книги может немного испугать. Я сам всегда настороженно отношусь к толстым книгам, потому что у всех нас не так уж много времени, которое мы можем посвятить изучению чего-то нового (и которое гораздо важнее при выборе книги, чем ее цена). Поэтому я воспользовался форматом, который предпочитаю в таких случаях — двойной книги.

**Двойная книга** — это две книги под одной обложкой. Первая — обычная повествовательная книга, предназначенная для чтения от корки до корки. Моя цель в этой книге — предоставить краткий обзор темы, достаточный для понимания, но не для практической работы. Объем этой части — не более пары сотен страниц, что вполне можно позволить себе прочесть полностью.

Вторая (бо́льшая по объему) книга представляет собой справочный материал, который предназначен не для подробного чтения (хотя некоторые люди так и делают), а для того, чтобы при необходимости обратиться к ней, как к справочнику. Одни читатели полностью прорабатывают первую книгу, чтобы получить общее представление о теме, а затем обращаются ко второй части через предметный указатель — в поисках только того материала, который их интересует. Другие читают справочный раздел так же, как и описательный. При разделении книги на две я хотел помочь вам в выборе материала, который можно пропустить и в который следует погрузиться как следует. Решение за вами.

Я также попытался сделать справочную часть в достаточной степени самостоятельной, так что если вы захотите узнать, например, о построении деревьев, то можете прочесть только соответствующую часть книги и получить вполне приличное представление о том, что следует делать, даже если вы подзабыли материал первой части. Таким образом, как только вы прочтете первую часть книги, она станет справочником, что очень удобно при практической работе, когда вам нужно найти информацию о конкретных деталях.

Основная причина, по которой книга получилась такой большой, — я просто не придумал, как сделать ее покороче. Одна из главных моих целей в этой книге заключается в предоставлении читателю обзора большого количества различных методов предметно-ориентированных языков. Есть книги о генерации кода, о метапрограммировании в Ruby и об использовании генераторов анализаторов. Я же стремился охватить все эти методы, чтобы вы могли лучше понять их сходства и различия. Все они играют определенную роль в более широком ландшафте, и моя цель — так организовать путешествие по этому ландшафту, чтобы вы ознакомились не только с отдельными пейзажами, но и с каждой из его частей достаточно подробно, чтобы иметь возможность тут же начать работу с описанными методами.

## О чем вы узнаете из этой книги

Я задумал эту книгу как руководство по различным видам предметно-ориентированных языков и подходам к их построению. Часто, начиная экспериментировать с предметно-ориентированными языками, люди используют какую-то одну технологию. Смысл этой книги — показать вам широкий спектр методов, чтобы вы могли оценить, какие из них лучше всего подходят в ваших обстоятельствах. Я представил подробную информацию и примеры реализации многих из этих методов. Естественно, я не могу показать вам все, что можно сделать, но для принятия первоначальных решений этого должно быть достаточно.

Из первых глав вы получите представление о том, что такое предметно-ориентированные языки, когда они могут пригодиться и какова их роль в сравнении с библиотеками. Вы узнаете, с чего начать при построении внешних и внутренних предметно-ориентированных языков. В главах о внешних предметно-ориентированных языках рассказывается о роли синтаксического анализатора, полезности генератора анализаторов и о способах применения анализаторов для синтаксического анализа внешнего предметно-ориентированного языка. В главах, посвященных внутренним предметно-ориентированным языкам, показано, как использовать языковые конструкции в стиле таких язы-

ков. Хотя это и не подскажет вам, как лучше всего использовать ваш конкретный язык, но поможет понять, как методы одного языка соотносятся с методами других языков.

В главах, посвященных генерации кода, изложены стратегии генерации кода. Имеется также глава с очень кратким обзором нового поколения инструментов (в большей части книги я ориентируюсь на методы, которые использовались на протяжении десятилетий).

## Для кого предназначена эта книга

Основная целевая аудитория этой книги — профессиональные разработчики программного обеспечения, которые рассматривают возможность построения предметно-ориентированного языка. Я представляю такого читателя как имеющего по крайней мере пару лет опыта программирования и хорошо знакомого с основными идеями проектирования программного обеспечения.

Те, кто серьезно занимаются проектированием языков, вероятно, в этой книге не найдут для себя много нового. Я, однако, надеюсь, что вы сочтете полезным использованный мною подход к изложению представленной в книге информации. Хотя в этой области проделана огромная работа, особенно в научной сфере, в мире профессионального программирования в данном вопросе непочатый край работы.

Первые несколько глав повествовательной части книги будут полезны всем, кого интересуют предметно-ориентированные языки и их применение. В этой повествовательной части представлен обзор используемых в данной области технологий.

## Это не книга о Java или C#

Как и в большинстве книг, которые я пишу, изложенные идеи в значительной степени зависят от языка программирования. Один из моих главных приоритетов — раскрыть общие принципы и модели, которые можно использовать с любым языком программирования, с которым вы будете иметь дело. А значит, идеи в книге должны быть ценными для вас, если вы используете любой современный объектно-ориентированный язык программирования.

Одним из потенциальных пробелов книги являются функциональные языки программирования. Хотя я думаю, что бо́льшая часть книги останется применимой и в этом случае, я не имею достаточного опыта работы с функциональными языками, чтобы понять, насколько их парадигма программирования изменит приведенные здесь советы. Книга несколько ограничена в случае процедурных языков (т.е. не объектно-ориентированных языков, таких как C), поскольку некоторые из описанных мною методов базируются на объектно-ориентированном подходе.

Хотя я описываю общие принципы, для того чтобы сделать это правильно, необходимы примеры применения конкретного языка программирования. При выборе языка для примеров основным критерием служила распространенность языка; в результате почти все примеры в этой книге написаны на Java или C#. Оба они широко используются и у обоих — C-образный синтаксис, управление памятью и библиотеки, позволяющие устранить многие неприятности. Я не утверждаю, что это наилучшие языки для написания предметно-ориентированных языков (в частности, потому, что лично я так не думаю), но они являются неплохими языками для представления описываемых мною общих концепций. Я пытался использовать оба языка в равной мере, склоняя чашу весов в пользу одного из них только тогда, когда он позволял упростить решение поставленной задачи. Я также старался избегать элементов языка, которые требуют слишком хорошего знания

синтаксиса, хотя это и труднодостижимый компромисс, так как умелое применение внутренних предметно-ориентированных языков часто предусматривает использование синтаксических особенностей базовых языков.

Существует несколько идей, которые требуют обязательного применения динамического языка и поэтому не могут быть показаны на Java или C#. В таких случаях я обращался к Ruby, поскольку это динамический язык, с которым я знаком лучше всего. Он также неплохо подходит для написания предметно-ориентированных языков. И вновь, несмотря на мои личные симпатии к этому языку, не стоит делать вывод, что рассмотренные методы неприменимы в других ситуациях.

Я должен отметить, что имеется много других языков программирования, пригодных для предметно-ориентированных языков, включая ряд языков, специально разработанных для упрощения создания внутренних предметно-ориентированных языков. Я не упоминаю их здесь, потому что не настолько много работал с ними, чтобы с уверенностью их вам рекомендовать. Однако не нужно рассматривать это заявление как мое отрицательное мнение о них.

В частности, одна из трудностей написания книги о предметно-ориентированных языках, не привязанной к конкретному языку программирования, состоит в том, что полезность многих методов очень сильно зависит от особенностей конкретного языка. Всегда следует помнить, что ваша языковая среда может серьезно сместить акценты по сравнению с обобщениями, которые я вынужден делать в этой книге.

## Чего не хватает в этой книге

Один из наиболее разочаровывающих моментов при написании книги — когда я понимаю, что пора остановиться. Мне понадобилось несколько лет, чтобы написать ее, и я верю, что эти годы потрачены не зря. Но я осознаю, что в книге имеется и много пробелов. Я хотел бы заполнить их, но на это потребуется слишком много времени. Мне представляется, что лучше иметь неполную, но опубликованную книгу, чем годами ждать выпуска полной книги, если таковая вообще возможна. Поэтому я упомяну главные пробелы, которые я вижу, но у меня нет времени на их заполнение.

Я уже упоминал об одном из них — о роли функциональных языков. Существует большой опыт создания предметно-ориентированных языков на современных функциональных языках на базе ML и/или Haskell, но в моей книге эта работа, по сути, проигнорирована. Это интересный вопрос — насколько знакомство с функциональными языками может повлиять на структуру изложенного материала?

Возможно, наиболее разочаровывающим пробелом для меня является отсутствие достойного обсуждения вопросов диагностики и обработки ошибок. Я помню, как в университете нас учили, что по-настоящему важной частью компиляторов является диагностика, так что я отлично понимаю, насколько важную тему я не раскрыл.

Моя любимая часть этой книги — раздел об альтернативных вычислительных моделях. Я мог бы написать гораздо больше, но время неумолимо. В конце концов я решил, что, хотя я написал меньше, чем мог бы, этого должно быть достаточно, чтобы вдохновить вас на самостоятельное изучение других книг.

## Справочник

Повествовательная часть книги имеет обычную структуру, но я думаю, что нужно немного подробнее рассказать о структуре справочного раздела. Я разбил его на ряд тем,

сгруппированных в главы, чтобы однотипные темы находились в одном месте. Моя цель — чтобы каждая тема была самостоятельным материалом (если вы прочли повествовательную часть книги, то сможете погрузиться в интересующую вас тему без необходимости обращаться к другому материалу). При наличии исключений я упоминаю об этом в начале соответствующей темы.

Большинство тем описаны как шаблоны. Цель шаблона — решение некоторой часто возникающей задачи. Так, если проблема формулируется как "Как структурировать синтаксический анализатор?", для ее решения можно воспользоваться двумя моделями — Delimiter-Directed Translation (213) и Syntax-Directed Translation (229).

О шаблонах для разработки программного обеспечения за последние пару десятилетий написано очень много, и разные авторы имеют разные точки зрения на них. Я считаю шаблоны полезными хотя бы потому, что они обеспечивают способ структурирования справочного раздела, как в данной книге. В повествовательной части говорится, что если необходимо выполнить анализ текста, то наиболее вероятными кандидатами являются две указанные модели; шаблоны же дают подробную информацию для выбора одной из них и начала ее реализации.

Хотя бо́льшая часть справочного раздела написана с использованием шаблонной структуры, я не применял ее абсолютно везде. Этот метод подходит не для всех разделов справочника. В ряде тем, таких как Nested Operator Expression (333), решение в действительности не оказывается центральным моментом темы, и тема не укладывается в структуру, используемую мною для шаблонов. Поэтому в подобных ситуациях я прибегаю к другому способу описания. Есть и иные случаи, которые трудно назвать шаблонами, такие как Macros (195) или BNF (237), но при этом использование шаблонной структуры оказалось хорошим способом описания. В целом я руководствовался шаблонной структурой, в частности отделение "как это работает" от "когда это использовать", похоже, вполне укладывается в описываемую мною концепцию.

## Шаблонная структура

Большинство авторов при описании шаблонов используют некоторый стандартный шаблон. Я столь же "шаблонен" и потому прибегаю к шаблонам, хотя они и не столь "шаблонны", как у других авторов. Мой шаблон впервые использован в [10].

Пожалуй, самым важным элементом моего шаблона является **название**. Одна из главных причин, по которым я использую шаблоны подобно элементам предметного указателя, — это помогает создать строгий словарь для обсуждения темы. Конечно, нет никакой гарантии, что этот словарь будет широко применяться, но по крайней мере он заставляет быть последовательным при написании книги и при этом служит для других отправной точкой, если, конечно, они пожелают его использовать.

Следующие два элемента — это **намерение** и **набросок**. Они кратко характеризуют шаблон в целом. Они также являются напоминанием о модели, так что, если вы точно знаете, что есть подходящий шаблон, но не помните его названия, эти элементы могут помочь вашей памяти. Намерение представляет собой одно-два предложения, а набросок — нечто более наглядное. Иногда я использую в качестве наброска диаграмму, иногда — фрагмент кода, т.е. то, что, как мне кажется, способно быстро передать сущность рассматриваемой модели. При использовании диаграммы я иногда использую UML, но при этом с радостью воспользуюсь чем угодно, если решу, что так смысл будет передан более наглядно.

Далее наступает очередь **резюме**, как правило, сосредоточивающегося на мотивационном примере. Это пара абзацев, назначение которых все то же — помочь людям получить обзор шаблона, прежде чем погрузиться в детали.

Два основных раздела описания шаблона — это "Как это работает" и "Когда это использовать". Порядок этих разделов достаточно произволен: если вы пытаетесь решить, следует ли использовать шаблон, можете прочесть только раздел "Когда это использовать". Однако весьма часто этот раздел не имеет смысла без знания, как работает данный шаблон.

Последнее разделы — с примерами. Хотя я делаю все, чтобы объяснить, как работает шаблон, в разделе "Как это работает", чтобы поставить точку, часто следует приводить конкретные примеры. Беда в том, что примеры кода опасны, потому что показывают только одно приложение шаблона, и можно подумать, что шаблон — это и есть приведенное приложение, а не общая концепция. Вы можете использовать один и тот же шаблон сто раз, причем всякий раз немного иначе, но у меня весьма ограниченное количество как бумаги, отпущенной на книгу, так и энергии для написания примеров. Таким образом, всегда помните, что картина гораздо шире, чем конкретика приведенного примера.

Все примеры преднамеренно очень просты и сфокусированы только на рассматриваемом шаблоне. Я использую простые и независимые примеры, потому что они соответствуют моей цели — сделать каждую главу справочника самостоятельной. Естественно, при практическом применении шаблона возникнет масса других вопросов, но простой пример, как мне кажется, дает по крайней мере шанс понять основные моменты. Более мощные примеры будут реалистичнее, но заставят вас иметь дело с массой вопросов, не относящихся к изучаемому шаблону. Моя цель — показать вам фрагменты головоломки, а уж собрать их вместе для ваших конкретных целей — это ваша проблема.

Это также означает, что еще одним приоритетом являлась понятность кода. Я не учитывал вопросы производительности, обработки ошибок и другие моменты, которые отвлекали бы от сущности шаблона.

Я старался также избегать кода, которому, по моему мнению, трудно следовать, даже если это идиомы использованного мною языка.

В описании ряда шаблонов будут отсутствовать те или иные разделы; в некоторых случаях это может быть раздел примеров, например когда пример к другому шаблону лучше (в таких ситуациях я указываю эти более подходящие примеры).

## Благодарности

Как обычно, когда я пишу книгу, в ее издании участвует много людей. И хотя на обложке указано мое имя, есть много людей, без которых книга была бы значительно менее качественной.

Моя первая благодарность — моей коллеге Ребекке Парсонс (Rebecca Parsons). Одна из моих проблем при написании этой книги заключалась в недостатке серьезного академического образования. Ребекка с ее опытом в теории языков программирования оказала мне огромную помощь. Кроме того, она — один из ведущих технических специалистов-практиков ThoughtWorks, так что она сочетает отличную академическую подготовку с большим практическим опытом. Ей хотелось (и, безусловно, она достаточно квалифицированна для этого) играть более важную роль в издании этой книги, но компания ThoughtWorks сочла ее слишком ценным работником. Я рад, что Ребекка все же смогла выкроить время для наших многочасовых бесед.

Когда дело доходит до рецензентов, автор всегда надеется (и отчасти боится) на рецензента, который прочтет книгу и найдет в ней все проблемные места. Мне посчастливилось познакомиться с Майклом Хангром (Michael Hunger), который замечательно сыг-

рал эту роль. С первых дней появления книги на моем сайте он начал бомбардировать меня сообщениями об ошибках и советами по их устранению, и поверьте, это было то, что нужно... Майкл также подталкивал меня к описанию методов с использованием статической типизации, в частности по отношению к статически типизированным таблицам символов (Symbol Table (177)). Он сделал множество дополнительных предложений, которых хватит еще на пару книг, и я надеюсь когда-нибудь их написать.

За несколько последних лет мы с коллегами Ребеккой Парсонс (Rebecca Parsons), Нилом Фордом (Neal Ford) и Ола Бини (Ola Bini) написали несколько учебных пособий по данному материалу. В процессе работы над ними было сформулировано множество идей, которые вошли в данную книгу.

Компания ThoughtWorks щедро дала мне много свободного времени, чтобы написать эту книгу. Я рад, что нашел компанию, которой удалось сделать так, чтобы мне хотелось оставаться активным членом ее команды.

У меня была сильная группа официальных экспертов, которые от корки до корки изучили эту книгу, нашли, как я надеюсь, все ошибки и предложили массу улучшений:

Дэвид Бок (David Bock)
Жилад Брака (Gilad Bracha)
Айно Корри (Aino Corry)
Свен Эффтинж (Sven Efftinge)
Эрик Эванс (Eric Evans)
Джей Филдс (Jay Fields)
Стив Фриман (Steve Freeman)
Брайан Готц (Brian Goetz)
Стив Хайес (Steve Hayes)
Клиффорд Хиз (Clifford Heath)
Майкс Хангр (Michael Hunger)
Дэвид Инг (David Ing)
Джереми Миллер (Jeremy Miller)
Рави Мохан (Ravi Mohan)
Тиранс Парр (Terance Parr)
Нат Прайс (Nat Pryce)
Крис Селлс (Chris Sells)
Натаниэль Шутта (Nathaniel Schutta)
Крег Тавернер (Craig Taverner)
Дэйв Томас (Dave Thomas)
Гленн Вандербург (Glenn Vanderburg)

Небольшое, но важное спасибо Дэвиду Ингу (David Ing) за название “Зоопарк DSL”.

Одна из приятных сторон должности редактора серии — возможность подобрать действительно хорошую команду авторов, которая помогает решать все вопросы и оттачивать идеи. Я особенно хочу поблагодарить Эллиотта Расти Харольда (Elliotte Rusty Harold) за его подробнейшие комментарии и анализ.

В качестве генераторов идей выступали многие мои коллеги из ThoughtWorks. Я хочу поблагодарить всех, кто разрешал мне копаться в своих проектах в течение последних нескольких лет. В них я нашел гораздо больше идей, чем могу описать, и я получал большое удовольствие от этого поиска.

Два человека сделали весьма ценные замечания по Интернету, на которые мне удалось отреагировать, прежде чем книга была передана в печать: Павел Бернхаузер (Pavel Bernhauser, “Mocky”) и Роман Яковенко (Roman Yakovenko, “tdyer”).

Спасибо всем сотрудникам издательства, которое опубликовало мою книгу. Особенно хочется отметить работу выпускающего редактора Грега Донча (Greg Doench) и главного редактора Джона Фуллера (John Fuller).

Дмитрий Кирсанов (Dmitry Kirsanov) превратил мой небрежный английский в язык, достойный страниц книги, а Алина Кирсанова (Alina Kirsanova) подготовила макет книги и ее предметный указатель.

## Ждем ваших отзывов!

Вы, читатель этой книги, и есть главный ее критик. Мы ценим ваше мнение и хотим знать, что было сделано нами правильно, что можно было сделать лучше и что еще вы хотели бы увидеть изданным нами. Нам интересны любые ваши замечания в наш адрес.

Мы ждем ваших комментариев и надеемся на них. Вы можете прислать нам бумажное или электронное письмо либо просто посетить наш сайт и оставить свои замечания там. Одним словом, любым удобным для вас способом дайте нам знать, нравится ли вам эта книга, а также выскажите свое мнение о том, как сделать наши книги более интересными для вас.

Отправляя письмо или сообщение, не забудьте указать название книги и ее авторов, а также свой обратный адрес. Мы внимательно ознакомимся с вашим мнением и обязательно учтем его при отборе и подготовке к изданию новых книг.

Наши электронные адреса:

E-mail: info@williamspublishing.com
WWW: http://www.williamspublishing.com

Наши почтовые адреса:

в России: 127055, г. Москва, ул. Лесная, д. 43, стр. 1
в Украине: 03150, Киев, а/я 152