Правила программирования

Как писать качественный код

Крис Циммерман

УДК 004.414.2
ББК 32.973.202

Ц61

The Rules of Programming: How to Write Better Code
Chris Zimmerman

Циммерман, Крис.

Ц61 Правила программирования. Как писать качественный код: [перевод с английского О. Перфильева]. — Алматы: Астана иностранная пресса, 2026. — 368 с. — (O’Reilly. Книги по программированию).

ISBN 978-601-12-7677-1

«Правила программирования» — практическое руководство по созданию качественного, понятного и поддерживаемого кода. Вместо абстрактной теории автор предлагает 21 правило, основанное на десятилетиях работы в Microsoft и студии Sucker Punch Productions, создавшей такие игровые проекты, как Sly Cooper, inFamous и Ghost of Tsushima. На многочисленных примерах автор показывает, как принимать инженерные решения, которые делают код проще, надежнее и удобнее для дальнейшего развития.

Книга помогает сформировать профессиональное мышление программиста и предлагает систему принципов, которые одинаково полезны как начинающим разработчикам, так и опытным инженерам.

УДК 004.414.2
ББК 32.973.202

ISBN 978-601-12-7677-1

Оглавление

Отзывы о книге «Правила программирования»

«Правила программирования» — это не только советы для новичков, но и уроки, которые могут научить кое-чему даже экспертов. Легкий и живой язык Циммермана доказывает, что книга может быть одновременно и занимательной, и поучительной.

Марк Серни,
главный системный архитектор PlayStation 4 и 5

«Правила программирования» дают ценные советы как начинающим, так и опытным программистам. Книга написана увлекательным стилем, а перечисленные Циммерманом правила представляют собой очень важный вклад в развитие практик программирования, и это в то время, когда технологии пронизывают почти все сферы бизнеса и общества.

Пол Доэрти,
исполнительный и технический директор компании Accenture

В этой книге описаны практичные и рабочие правила, которые помогут повысить свой уровень любому программисту. Мне повезло — в начале своей карьеры я узнал эти правила от самого Криса, и я успешно применял их в самых разных областях разработки. Теперь с ними можете познакомиться и вы.

Крис Бентцель,
директор по разработке программного обеспечения, Boston Dynamics

Предисловие

Здравствуйте! Перед вами «Правила программирования» — свод легких для запоминания и простых в использовании советов, которые помогут вам научиться составлять более продуманные и качественные программы. Программирование — сложная работа, но правила делают ее чуточку проще.

Перед началом вот вам несколько советов:

- Все Правила самодостаточны. Если вы увидели в оглавлении какое-то Правило, которое показалось вам интересным, и хотите сразу же перейти в середину книги, — пожалуйста, переходите. Такой стиль чтения полностью поддерживается.
- Тем не менее я бы рекомендовал начать с Правила 1 «Максимально просто, но не слишком просто». Оно задает хороший тон для всех последующих Правил.
- Примеры в книге написаны на языке C++. Если вы программируете на Python или JavaScript, то вам стоит для начала ознакомиться с Приложением А («Чтение C++ для программистов на Python») или Приложением Б («Чтение C++ для программистов на JavaScript»). Это своего рода «Розеттский камень», помогающий перевести конструкции C++ в привычные для вас концепции. Если вы работаете с каким-то другим языком и находите примеры на C++ трудными для восприятия, рекомендую посетить замечательный сайт Rosetta Code.
- Тем, кто программирует на C++, следует иметь в виду, что некоторые фрагменты кода в примерах я упростил для удобства тех, кто не работает с C++. Например, в некоторых местах используются знаковые целочисленные типы, тогда как в типичной программе на C++, скорее всего, использовались бы беззнаковые типы. Также я отключил предупреждения о неявном преобразовании между знаковыми и беззнаковыми типами. Кроме того, в примерах не указывалась явно директива «using std», хотя компилировались они с ней. Я исключил ее, чтобы не загромождать код отвлекающими внимание ссылками «std::».
- И наконец, когда я ссылаюсь на какое-то конкретное правило из этой книги, я пишу слово «Правило» с заглавной буквы. Если вы видите слово «правило», написанное со строчной буквы, — это просто

обычное правило, а не какое-то формально описанное в этой книге. Без заглавной буквы могла бы возникнуть путаница, и, надеюсь, вы меня за это простите.

Надеюсь, книга вам понравится и вы откроете для себя несколько полезных мыслей, которые помогут вам отточить свои навыки программирования.

Girls Who Code

Все авторские отчисления от этой книги направляются организации Girls Who Code, прилагающей все усилия ради того, чтобы помочь девушкам открыть увлекательный и полезный мир программирования. Когда я заканчивал колледж, более трети выпускников по компьютерным наукам составляли женщины; сегодня же их доля составляет около одной пятой. Я считаю, что при более сбалансированном гендерном соотношении все мы только выиграли бы. Надеюсь, что вы тоже так считаете. Поддержка Girls Who Code, будь то финансовая или в форме волонтерства, — это шаг к воплощению надежды в реальность.

Условные обозначения

В этой книге используются следующие типографские условные обозначения:

Курсив

Используется для новых терминов, URL, адресов электронной почты, имен файлов и расширения файлов.

`Моноширинный шрифт`

Используется для программного кода, а также в тексте для обозначения таких элементов программы, как имена переменных и функций, базы данных, типы данных, переменные среды, операторы и ключевые слова.

Использование примеров кода

Дополнительные материалы (примеры кода, упражнения и т. д.) можно скачать по адресу: *https://oreil.ly/rules-of-programming-code*.

Если у вас возникнут какие-то технические вопросы или если вы столкнетесь с какой-либо проблемой при использовании примеров кода, то вы можете отправить сообщение по адресу электронной почты: *bookquestions@oreilly.com*.

Эта книга призвана помочь вам в вашей работе. Как правило, любой упоминаемый в книге фрагмент кода вы можете свободно использовать в своих

программах и в своей документации. Вам не нужно обращаться к нам за разрешением, если вы не воспроизводите значительную часть кода. Например, если вы захотите включить в свою работу несколько строк из примера, то получать разрешение для этого не нужно. Но если вы хотите продавать или распространять примеры из книг O'Reilly, то для этого требуется разрешение. Если вы отвечаете на какой-то вопрос и приводите пример из этой книги или ссылку на эту книгу, то разрешения не требуется, но если вы собираетесь использовать большое количество примеров из этой книги в документации к своему продукту, то для этого нужно получить разрешение.

Мы признательны всем, кто указывает авторство, но обычно не требуем такого указания. При этом, как правило, указание авторства подразумевает упоминание источника, автора, издателя и ISBN, например: «"Правила программирования", Крис Циммерман (O'Reilly). Copyright 2023 Крис Циммерман, 978-1-098-13311-5».

Если вы считаете, что ваше использование примеров кода выходит за рамки добросовестного использования или данного выше разрешения, пожалуйста, свяжитесь с нами по адресу: *permissions@oreilly.com*.

O'Reilly Online Learning

На протяжении более 40 лет O'Reilly Media предоставляет знания и навыки в области технологий и бизнеса, а также предлагает материалы, помогающие компаниям добиваться успеха.

Наша уникальная сеть экспертов и новаторов делится своим опытом и знаниями в книгах, статьях и на нашей онлайн-платформе обучения. Онлайн-платформа O'Reilly предоставляет по запросу доступ к живым курсам, углубленным учебным программам, интерактивным средам программирования, а также обширной коллекции текстов и видео от O'Reilly и более чем 200 других издателей. Дополнительная информация указана на странице по адресу: h*ttps://oreilly.com*.

Как с нами связаться

Все касающиеся этой книги вопросы и комментарии направляйте издателю:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (США и Канада)
707-829-0515 (международные и местные звонки)
707-829-0104 (факс)

У этой книги есть своя веб-страница, на которой публикуются исправления, примеры и дополнительная информация: *https://oreil.ly/rules-of-programming*.

Если вы хотите поделиться своими впечатлениями, высказать комментарии или задать вопросы автору, зайдите на сайт книги *The Rules of Programming*, где вы найдете нужные указания. Также вы можете написать по адресу: *bookquestions@oreilly.com*, чтобы оставить свои отзывы или задать технические вопросы по поводу этой книги.

Новости и информацию о наших книгах и курсах можно найти по адресу: *https://oreilly.com*.

Благодарности

Прежде всего я благодарю свою прекрасную и талантливую жену Лору, которая убеждала меня тратить время на эту книгу, а не на всякие другие полезные дела, какими я мог бы заняться.

Огромная благодарность всем, кто помогал развивать описанные в этой книге Правила. Среди них — все бывшие и нынешние программисты Sucker Punch, осознанно или неосознанно внесшие свой вклад в книгу; особенно же стоит отметить следующих: Апурва Бансал, Крис Хайдорн, Дэвид Майер, Эрик Блэк, Эван Кристенсен, Джеймс Макнилл, Джасмин Патри, Нейт Слоттоу, Мэтт Дурасофф, Майк Гаффни, Ранджит Раджагопалан, Роб Макдэниел, Сэм Холли, Шон Смит, Уэс Грэндмон и Уильям Росситер.

Я говорю спасибо также всем, кто не связан с Sucker Punch и кто смог посмотреть на книгу свежим взглядом со стороны: Адаму Барру, Андреасу Фредрикссону, Колину Брайару, Дэвиду Оливеру, Максу Шуберту, Майку Гутманну и Сету Файну.

Отдельная особая благодарность отважным читателям, одолевшим все без исключения Правила: Адриану Бентли, Биллу Рокенбеку, Яну Миксовски и Джулиену Мерсерону. Я официально обязан отплатить вам ответной услугой.

И наконец, спасибо всей команде O'Reilly, которая терпеливо поддерживала меня во всех моих неуклюжих попытках написать эту книгу: Чарльзу Румелиотису, Грегори Хайману, Либби Джеймс, Мэри Треселер, Саре Хантер, Сьюзанн Хьюстон и особенно Саре Грей, оказавшей всем вам огромную услугу и выкинувшей из книги все самые несмешные шутки, которые я так и норовил в нее добавить.

История Правил

«Правила программирования» были созданы от отчаяния.

Около десяти лет я руководил командами программистов в Microsoft, а в 1997 году стал соучредителем игровой компании Sucker Punch. Обе компании добились успеха во многом благодаря своей способности привлекать первоклассных программистов и создавать из них великолепные коллективы. Sucker Punch может по праву похвастаться чередой успешных игр, которые выпускала на протяжении 25 лет. Прежде всего стоит отметить три игры из серии Sly Cooper, рассчитанных на детей всех возрастов и позволявших им прикоснуться к полной приключений жизни енота-вора Слая Купера и его друзей. Далее идут пять игр из серии inFamous, в которых игрокам предоставлялся выбор — воспользоваться суперсилами во благо или во зло. И, наконец, наше величайшее творение — это игра Ghost of Tsushima, в которой игроки примеряют на себя роль одинокого самурая, сражающегося против вторгшихся в Японию в 1274 году захватчиков*.

Как в Microsoft, так и в Sucker Punch важной составляющей стратегии найма был прием на работу молодых умных программистов с последующим их обучением профессиональным методам. Практика эта, несомненно, показала свой успех, но она же порождала и особую разновидность разочарования.

Раз за разом я сталкивался с одной и той же проблемой. К нам в команду приходил очередной новый программист, часто только что окончивший колледж. Я проверял какую-нибудь новую функцию, которую он планировал добавить в код — обычно для решения очень простой проблемы, — и оказывалось, что он пытался написать код, решающий гораздо более крупную проблему, которая включала в себя ту самую простую и конкретную задачу в качестве частного случая.

Но нам вовсе не нужно было решать ту самую большую проблему, по крайней мере не в тот момент! Постоянно оказывалось, что решение большой проблемы — это посредственное решение для той простой стоявшей

---

* Самые наблюдательные из вас, вероятно, заметили, что я не упомянул первую игру Sucker Punch, а именно Rocket: Robot on the Wheels. Это потому что в нее играли немногие; если вы находитесь в числе избранных, поздравляю вас.

перед нами задачи. Решение более сложное в использовании, сложное для понимания и потенциально содержащее гораздо большее количество ошибок. Но сколько бы я ни повторял на ревизии кода* о том, что программистам нужно браться только за те задачи, которые им по плечу и которые они понимают, новички продолжали делать всё по-своему.

От отчаяния я пошел на решительные меры. «Ну что ж, — сказал я. — Вот вам новое правило. Пока у вас нет трех примеров решения конкретной проблемы, вы не имеете права придумывать обобщенное решение».

К моему удивлению и радости, это сработало! Превратив общие рассуждения в конкретное правило с четкими критериями, я донес-таки до своих подчиненных нужную мысль. Конечно, большинство наших новых программистов все равно хотя бы раз совершали подобную ошибку чрезмерного обобщения, но сформулированное мной правило помогало им не повторять ее. И оно же помогало им понимать, когда действительно нужно было переходить к обобщению задачи. Меньше трех конкретных примеров? Не обобщай. Три или больше? Пора придумывать что-то большее.

Правило сработало, потому что его было легко запомнить, а ситуации, в которых оно применялось, было легко распознать. Выходя за границы четко определенной текущей задачи, программисты вспоминали правило, подсчитывали, сколько примеров такой задачи они уже выполнили, и решали, что им делать дальше — обобщать или решать конкретную задачу. В результате код у них выходил лучше.

Со временем мы определили и другие важные элементы философии Sucker Punch, которые можно было свести к легким для запоминания фразам, точнее, афоризмам. У афоризмов (или пословиц), то есть коротких и емких высказываний, заключающих в себе некую существенную истину, долгая история. Вы и сами можете с ходу назвать несколько афоризмов по памяти. Да что там — вы бы с легкостью вспомнили несколько пословиц только про птиц. Вот несколько примеров:

- Цыплят по осени считают**.
- Лучше синица в руках, чем журавль в небе.
- Слово не воробей — вылетит, не поймаешь.
- Не клади все яйца в одну корзину.

Афоризмы приживаются, потому что они работают. На современном жаргоне их можно было бы назвать «вирусными». Они «заряжали» людей

---

* Любой код в рамках какого-либо проекта Sucker Punch всегда проходит ревизию. Подробнее об этом говорится в Правиле 6.

** Похожее высказывание встречается еще в «Новых сонетах и прелестных памфлетах» Томаса Хауэлла 1570 года: «Не считай своих цыплят, покуда они не вылупились…» Небольшой пример долговечности хорошего афоризма.

крупицами мудрости на протяжении тысячелетий*. Неудивительно, что они оказались эффективным способом «заразить» новых членов команды философией программирования Sucker Punch.

Итак, шаг за шагом некогда единственное правило разрослось в целый свод описанных в этой книге «Правил программирования». Они отражают многие важнейшие аспекты принятой в Sucker Punch культуры программирования, то есть те принципы, которые, по нашему мнению, привели нас к успеху, и те идеи, которые новые члены коллектива должны усвоить для эффективной работы. И их время от времени нужно вспоминать даже таким опытным программистам, как я!

Каждая из следующих глав описывает одно из Правил со множеством примеров, иллюстрирующих стоящую за ним идею. Прочитав главу, вы получите четкое представление о каком-то конкретном практическом методе программирования и о ситуациях, в которых применим этот метод.

Окажутся ли Правила столь же «вирусными» в формате книги? Давайте проверим.

---

* Сам термин *афоризм* ввёл в обращение древнегреческий целитель Гиппократ около 400 года до н. э. Строго говоря, это было слово Ἀφορισμός. Так назывался его свод правил диагностики и лечения, некоторые из которых сохраняют свою силу и спустя тысячелетия, как, например, афоризм 13 из раздела 6: «Приступ чихания излечивает от икоты». Что абсолютно правда.

Как не соглашаться с Правилами

Надеюсь, вы не станете слепо следовать каждому описанному в этой книге 21 Правилу.

Если вы вдруг заметите, что просто вежливо киваете, читая мои Правила с примерами, и говорите: «Ну да, это логично; примеры знакомы, у меня тоже была такая мысль, просто я описывал ее другими словами», — ну что ж, я сочту это провалом.

Я хочу вам дать пищу для размышлений. В идеале хотелось бы, чтобы вы получили одно-два озарения. Возможно, я дам какое-то название тому смутному ощущению, которое у вас уже было, или покажу четкий пример того, чему вы не могли найти определение. Может, я даже предложу вам нечто новое для осмысления.

Но также вероятно, что вы столкнетесь с одной-двумя идеями, с которыми не согласитесь. Вы можете даже подумать, что в чем-то я совершенно не прав и что какое-то из Правил — это плохой совет.

И это прекрасно! Найти Правило, с которым вы не соглашаетесь, — это повод глубже задуматься. Отвергать его тут же, не задумываясь лишний раз, было бы ошибкой.

Я обещаю, что каждое из описанных здесь правил имеет свой смысл, но не утверждаю, что это абсолютный закон. Одно и то же правило может быть хорошим для нас и одновременно неподходящим для вас. Поняв почему, вы лучше поймете себя и свою философию программирования. Вы осознаете различия между компанией Sucker Punch и вашей собственной командой; вы поймете, почему эти различия делают какое-то Правило важной частью нашей культуры, но почему оно плохо совместимо с вашей культурой. В Правилах используются примеры из видеоигр, которые создавала студия Sucker Punch и в которых отражены некоторые наши особенности программирования в сфере видеоигр. Многие другие аспекты затрагиваются в последней главе «Заключение. Настраивайте правила под себя».

Когда я сталкиваюсь с каким-то утверждением о философии программирования, которое противоречит моему собственному опыту, я стараюсь осознать его с помощью следующих шагов:

1. Найти в высказывании не только недостатки, но и какое-то зерно истины. Возможно, я не соглашаюсь с утверждением, исходя из каких-то своих личных соображений. При каких обстоятельствах это утверждение было бы верным?

2. Рассмотреть проблему с другой стороны. При каких обстоятельствах моя точка зрения была бы ошибочной? Какие условия меняют истинность утверждения?

3. Вспомнить о том, что обстоятельства меняются. Утверждение может показаться вам неподходящим именно сейчас, но оказаться верным для другого вашего проекта. Так вы определите, в какой ситуации ваша философия должна будет измениться; в следующий раз вы быстрее распознаете ситуацию, в которой можно было бы и прислушаться к данному утверждению.

Я проходил через этот процесс много раз, и один из примеров иной ситуации — это так называемая «разработка через тестирование» (TDD, test-driven development). Она не сочетается с принятыми у нас в Sucker Punch методами разработки, но мы понимаем, что в каких-то случаях она оправданна. Читая Правила и размышляя над ними, вы тоже поймете, почему наши конкретные условия не подходят для TDD, но это вовсе не значит, что условия эти не могут поменяться. Мы внимательно следим за ситуацией; если что-то изменится, то изменится и наша философия.

Итак, я надеюсь, что вы найдете что-то полезное даже в тех Правилах, с которыми не согласны… но я пойму вас и в том случае, если вы решите последовать другому принципу, описанному Дороти Паркер:

Этот роман не нужно легкомысленно отбрасывать. Его нужно швырнуть прочь изо всех сил.

Если так, советую целиться во что-то мягкое.

Правило 1

Максимально просто, но не слишком просто

Программировать трудно.

Думаю, вы и сами уже пришли к такому выводу. Любой, кто берет в руки и читает книгу «Правила программирования», скорее всего:

- Умеет программировать, по крайней мере немного.
- Испытывает раздражение от того, что это не так просто.

Программировать трудно по многим причинам, и в свое время было разработано немало стратегий с целью облегчить это занятие. В этой книге рассматриваются примеры распространенных ошибок и Правила, помогающие их избежать; все они основаны на моем долгом личном опыте совершения похожих ошибок и попыток исправить ошибки других.

В этих Правилах прослеживается общая закономерность, некая объединяющая большинство из них сквозная тема. Лучше всего ее описывает цитата Альберта Эйнштейна, в которой говорится о принципах работы физика-теоретика: «Максимально просто [как можно проще], но не слишком просто»*. Этим Эйнштейн хотел сказать, что лучшая физическая теория — это самая простая из тех, что полностью описывают все наблюдаемые явления.

Если применить эту идею к сфере программирования, то можно утверждать, что лучшее решение любой задачи — это самое простое решение, которое удовлетворяет всем требованиям этой задачи. Лучший код — это самый простой код.

Предположим, что вы пишете код для подсчета количества битов в целочисленной переменной. Сделать это можно разными способами. Можно вос-

---

* Он почти наверняка не говорил этой фразы именно в таком виде. Потомки оказали Эйнштейну услугу, отточив его афоризмы. В письменных источниках можно найти только следующее наиболее близкое к этой фразе высказывание: «Нельзя отрицать, что высшая цель любой теории — сделать нередуцируемые базовые элементы как можно более простыми и немногочисленными, не отказываясь от адекватного представления любых опытных данных». В целом почти то же самое, просто не так емко. К тому же оригинальная цитата Эйнштейна немного длинновата для названия Правила.

пользоваться «битовыми трюками»* — например, обнулять по одному биту за раз и подсчитывать количество обнуленных битов:

```
int countSetBits(int value)
{
    int count = 0;
    while (value)
    {
        ++count;
        value = value & (value - 1);
    }
    return count;
}
```

Можно также реализовать эту идею без циклов, с использованием битовых сдвигов и масок для параллельного подсчета битов:

```
int countSetBits(int value)
{
    value = ((value & 0xaaaaaaaa) >> 1) + (value & 0x55555555);
    value = ((value & 0xcccccccc) >> 2) + (value & 0x33333333);
    value = ((value & 0xf0f0f0f0) >> 4) + (value & 0x0f0f0f0f);
    value = ((value & 0xff00ff00) >> 8) + (value & 0x00ff00ff);
    value = ((value & 0xffff0000) >> 16) + (value & 0x0000ffff);
    return value;
}
```

А можно просто написать самый очевидный код из возможных:

```
int countSetBits(int value)
{
    int count = 0;
    for (int bit = 0; bit < 32; ++bit)
    {
        if (value & (1 << bit))
            ++count;
    }
```

* Прошу прощения у всех, кто не программирует на C++, за битовые операции в следующих трех примерах. Обещаю, что в остальной части книги будет минимальное количество побитовых операций.