

что мы умеем рисовать. Таким образом, умение отличать чистый код от грязного еще не означает, что вы умеете писать чистый код!

Чтобы написать чистый код, необходимо сознательно применять множество приемов, руководствуясь приобретенным усердным трудом чувством «чистоты». Ключевую роль здесь играет «чувство кода». Одни с этим чувством рождаются. Другие работают, чтобы развить его. Это чувство не только позволяет отличить хороший код от плохого, но и демонстрирует стратегию применения наших навыков для преобразования плохого кода в чистый код.

Программист без «чувства кода» посмотрит на грязный модуль и распознает беспорядок, но понятия не имеет, что с ним делать. Программист с «чувством кода» смотрит на грязный модуль и видит различные варианты и возможности. «Чувство кода» поможет ему выбрать лучший вариант и спланировать последовательность преобразований, сохраняющих поведение программы и приводящих к нужному результату.

Короче говоря, программист, пишущий чистый код, — это художник, который проводит пустой экран через серию преобразований, пока он не превратится в элегантно запрограммированную систему.

Что такое «чистый код»?

Вероятно, сколько существует программистов, столько найдется и определений. Поэтому я спросил у некоторых известных, чрезвычайно опытных программистов, что они думают по этому поводу.

БЬЁРН СТРАУСТРУП, СОЗДАТЕЛЬ C++ И АВТОР КНИГИ «THE C++ PROGRAMMING LANGUAGE»

Я люблю, чтобы мой код был элегантным и эффективным. Логика должны быть достаточно прямолинейной, чтобы ошибкам было трудно спрятаться; зависимости — минимальными, чтобы упростить сопровождение; обработка ошибок — полной в соответствии с выработанной стратегией; а производительность — близкой к оптимальной, чтобы не искушать людей загрязнять код беспринципными оптимизациями. Чистый код хорошо решает одну задачу.



Бьёрн использует слово «элегантный». Хорошее слово! Словарь в моем Mac-Book® выдает следующие определения: доставляющий удовольствие своим изяществом и стилем; сочетающий простоту с изобретательностью. Обратите внимание на оборот «доставляющий удовольствие». Очевидно, Бьёрн считает,

что чистый код приятно читать. При чтении чистого кода вы улыбаетесь, как при виде искусно сделанной музыкальной шкатулки или хорошо сконструированной машины.

Бьёрн также упоминает об эффективности — притом дважды. Наверное, никого не удивят эти слова, произнесенные изобретателем C++, но я думаю, что здесь кроется нечто большее, чем простое стремление к скорости. Напрасные траты процессорного времени неэлегантны, они не радуют глаз. Также обратите внимание на слово «искушение», которым Бьёрн описывает последствия неэлегантности. В этом кроется глубокая истина. Плохой код *искушает*, способствуя увеличению беспорядка! Когда другие программисты изменяют плохой код, они обычно делают его еще хуже.

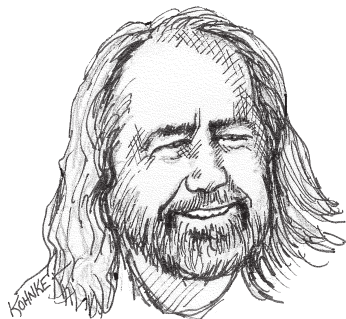
Прагматичные Дэйв Томас (Dave Thomas) и Энди Хант (Andy Hunt) высказали ту же мысль несколько иначе. Они сравнили плохой код с разбитыми окнами¹. Здание с разбитыми окнами выглядит так, словно никому до него нет дела. Поэтому люди тоже перестают обращать на него внимание. Они равнодушно смотрят, как на доме появляются новые разбитые окна, а со временем начинают сами бить их. Они уродуют фасад дома надписями и устраивают мусорную свалку. Одно разбитое окно стало началом процесса разложения.

Бьёрн также упоминает о необходимости полной обработки ошибок. Это одно из проявлений внимания к мелочам. Упрощенная обработка ошибок — всего лишь одна из областей, в которых программисты пренебрегают мелочами. Утечка — другая область, состоящая гонки — третья, непоследовательный выбор имен — четвертая... Суть в том, что чистый код уделяет пристальное внимание мелочам.

В завершение Бьёрн говорит о том, что чистый код хорошо решает одну задачу. Не случайно многие принципы проектирования программного обеспечения сводятся к этому простому наставлению. Писатели один за другим пытаются донести эту мысль. Плохой код пытается сделать слишком много всего, для него характерны неясные намерения и неоднозначность целей. Для чистого кода характерна целенаправленность. Каждая функция, каждый класс, каждый модуль фокусируются на конкретной цели, не отвлекаются от нее и не загрязняются окружающими подробностями.

**ГРЭДИ БУЧ, АВТОР КНИГИ
«OBJECT ORIENTED ANALYSIS
AND DESIGN WITH APPLICATIONS»**

Чистый код прост и прямолинеен. Чистый код читается, как хорошо написанная проза. Чистый код никогда не затемняет намерения проектировщика; он полон четких абстракций и простых линий передачи управления.



¹ <http://www.pragmaticprogrammer.com/booksellers/2004-12.html>.

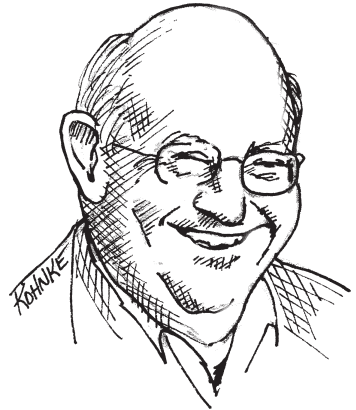
Грэди частично говорит о том же, о чем говорил Бьёрн, но с точки зрения *удобочитаемости*. Мне особенно нравится его замечание о том, что чистый код должен читаться, как хорошо написанная проза. Вспомните какую-нибудь хорошую книгу, которую вы читали. Вспомните, как слова словно исчезали, заменяясь зрительными образами! Как кино, верно? Лучшее! Вы словно видели персонажей, слышали звуки, испытывали душевное волнение и сопереживали героям.

Конечно, чтение чистого кода никогда не сравнится с чтением «Властелина колец». И все же литературная метафора в данном случае вполне уместна. Чистый код, как и хорошая повесть, должен наглядно раскрыть интригу решаемой задачи. Он должен довести эту интригу до высшей точки, чтобы потом читатель воскликнул: «Ага! Ну конечно!», когда все вопросы и противоречия благополучно разрешатся в откровении очевидного решения.

На мой взгляд, использованный Грэди оборот «четкая абстракция» представляет собой очаровательный оксюморон! В конце концов, слово «четкий» почти всегда является синонимом для слова «конкретный». В словаре моего MacBook приведено следующее определение слова «четкий»: *краткий, решительный, фактический, без колебаний или лишних подробностей*. Несмотря на кажущееся смысловое противоречие, эти слова несут мощный информационный посыл. Наш код должен быть фактическим, а не умозрительным. Он должен содержать только то, что необходимо. Читатель должен видеть за кодом нашу решительность.

«БОЛЬШОЙ» ДЭЙВ ТОМАС, ОСНОВАТЕЛЬ ОТI, КРЕСТНЫЙ ОТЕЦ СТРАТЕГИИ ECLIPSE

Чистый код может читаться и совершенствоваться другими разработчиками, кроме его исходного автора. Для него написаны модульные и приемочные тесты. В чистом коде используются содержательные имена. Для выполнения одной операции в нем используется один путь (вместо нескольких разных). Чистый код обладает минимальными зависимостями, которые явно определены, и четким, минимальным API. Код должен быть грамотным, потому что в зависимости от языка не вся необходимая информация может быть четко выражена в самом коде.



Большой Дэйв разделяет стремление Грэди к удобочитаемости, но с одной важной особенностью. Дэйв утверждает, что чистота кода упрощает его доработку другими людьми. На первый взгляд это утверждение кажется очевидным, но его важность трудно переоценить. В конце концов, код, который легко читается, и код, который легко изменяется, — не одно и то же.

Дэйв связывает чистоту с тестами! Десять лет назад это вызвало бы множество недоуменных взглядов. Однако методология разработки через тестирование оказала огромное влияние на нашу отрасль и стала одной из самых фундамен-

тальных дисциплин. Дэйв прав. Код без тестов не может быть назван чистым, каким бы элегантным он ни был и как бы хорошо он ни читался.

Дэйв использует слово «минимальный» дважды. Очевидно, он отдает предпочтение компактному коду перед объемистым кодом. В самом деле, это положение постоянно повторяется в литературе по программированию от начала ее существования. Чем меньше, тем лучше.

Дэйв также говорил, что код должен быть *грамотным*. Это ненавязчивая ссылка на концепцию «грамотного программирования» Дональда Кнута [Knuth92]. Итак, код должен быть написан в такой форме, чтобы он хорошо читался людьми.

МАЙКЛ ФИЗЕРС, АВТОР КНИГИ «WORKING EFFECTIVELY WITH LEGACY CODE»

Я мог бы перечислить все признаки, присущие чистому коду, но существует один важнейший признак, из которого следуют все остальные. Чистый код всегда выглядит так, словно его автор над ним тщательно потрудился. Вы не найдете никаких очевидных возможностей для его улучшения. Все они уже были продуманы автором кода. Попытавшись представить возможные усовершенствования, вы снова придете к тому, с чего все началось: вы рассматриваете код, тщательно продуманный и написанный настоящим мастером, небезразличным к своему ремеслу.



Всего одно слово: тщательность. На самом деле оно составляет тему этой книги. Возможно, ее название стоило снабдить подзаголовком: «Как тщательно работать над кодом».

Майкл попал в самую точку. Чистый код — это код, над которым тщательно поработали. Кто-то не пожалел своего времени, чтобы сделать его простым и стройным. Кто-то уделил должное внимание всем мелочам и относился к коду с душой.

РОН ДЖЕФФРИС, АВТОР КНИГ «EXTREME PROGRAMMING INSTALLED» И «EXTREME PROGRAMMING ADVENTURES IN C#»

Карьера Рона началась с программирования на языке Fortran. С тех пор он писал код практически на всех языках и на всех компьютерах. К его словам стоит прислушаться.



За последние коды я постоянно руководствуюсь «правилами простого кода», сформулированными Бекон. В порядке важности, простой код:

- проходит все тесты;
- не содержит дубликатов;
- выражает все концепции проектирования, заложенные в систему;
- содержит минимальное количество сущностей: классов, методов, функций и т. д.

Из всех правил я уделяю основное внимание дублированию. Если что-то делается в программе снова и снова, это свидетельствует о том, что какая-то мысленная концепция не нашла представления в коде. Я пытаюсь понять, что это такое, а затем пытаюсь выразить идею более четко.

Выразительность для меня прежде всего означает содержательность имен. Обычно я провожу переименования по несколько раз, пока не остановлюсь на окончательном варианте. В современных средах программирования — таких, как Eclipse — переименование выполняется легко, поэтому изменения меня не беспокоят. Впрочем, выразительность не ограничивается одними лишь именами. Я также смотрю, не выполняет ли объект или метод более одной операции. Если это объект, то его, вероятно, стоит разбить на два и более объекта. Если это метод, я всегда применяю к нему прием «извлечения метода»; в итоге у меня остается основной метод, который более четко объясняет, что он делает, и несколько подметодов, объясняющих, как он это делает.

Отсутствие дублирования и выразительности являются важнейшими составляющими чистого кода в моем понимании. Даже если при улучшении грязного кода вы будете руководствоваться только этими двумя целями, разница в качестве кода может быть огромной. Однако существует еще одна цель, о которой я также постоянно помню, хотя объяснить ее будет несколько сложнее.

После многолетней работы мне кажется, что все программы состоят из очень похожих элементов. Для примера возьмем операцию «найти элемент в коллекции». Независимо от того, работаем ли мы с базой данных, содержащий информацию о работниках, или хеш-таблицей с парами «ключ-значение», или массивом с однотипными объектами, на практике часто возникает задача извлечь конкретный элемент из этой коллекции. В подобных ситуациях я часто инкапсулирую конкретную реализацию в более абстрактном методе или классе. Это открывает пару интересных возможностей.

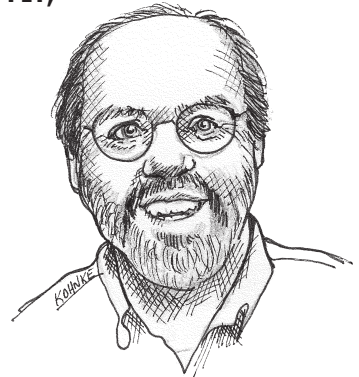
Я могу определить для нужной функциональности какую-нибудь простую реализацию (например, хеш-таблицу), но поскольку все ссылки прикрыты моей маленькой абстракцией, реализацию можно в любой момент изменить. Я могу быстро двигаться вперед, сохраняя возможность внести изменения позднее.

Кроме того, абстракция часто привлекает мое внимание к тому, что же «действительно» происходит в программе, и удерживает меня от реализации поведения коллекций там, где в действительности достаточно более простых способов получения нужной информации. Сокращение дублирования, высокая выразительность и раннее построение простых абстракций. Все это составляет чистый код в моем понимании.

В нескольких коротких абзацах Рон представил сводку содержимого этой книги. Устранение дублирования, выполнение одной операции, выразительность, простые абстракции. Все на месте.

УОРД КАННИНГЕМ, СОЗДАТЕЛЬ WIKI, СОЗДАТЕЛЬ FIT, ОДИН ИЗ СОЗДАТЕЛЕЙ ЭКСТРЕМАЛЬНОГО ПРОГРАММИРОВАНИЯ. ВОДХНОВИТЕЛЬ НАПИСАНИЯ КНИГИ «DESIGN PATTERNS». ДУХОВНЫЙ ЛИДЕР SMALLTALK И ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПОДХОДА. КРЕСТНЫЙ ОТЕЦ ВСЕХ, КТО ТЩАТЕЛЬНО ОТНОСИТСЯ К НАПИСАНИЮ КОДА.

Вы работаете с чистым кодом, если каждая функция делает примерно то, что вы ожидали. Код можно назвать красивым, если у вас также создается впечатление, что язык был создан специально для этой задачи.



Подобные заявления — отличительная способность Уорда. Вы читаете их, киваете головой и переходите к следующей теме. Это звучит настолько разумно, настолько очевидно, что не выглядит чем-то глубоким и мудрым. Вроде бы все само собой разумеется. Но давайте присмотримся повнимательнее.

«...примерно то, что вы ожидали». Когда вы в последний раз видели модуль, который делал примерно то, что вы ожидали? Почему попадающиеся нам модули выглядят сложными, запутанными, приводят в замешательство? Разве они не нарушают это правило? Как часто вы безуспешно пытались понять логику всей системы и проследить ее в том модуле, который вы сейчас читаете? Когда в последний раз при чтении кода вы кивали головой так, как при очевидном заявлении Уорда?

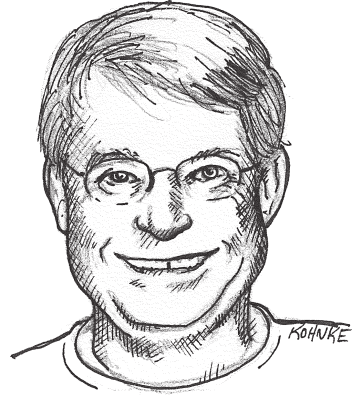
Уорд считает, что чтение чистого кода вас совершенно не удивит. В самом деле, оно даже не потребует от вас особых усилий. Вы читаете код, и он делает примерно то, что вы ожидали. Чистый код очевиден, прост и привлекателен. Каждый модуль создает условия для следующего. Каждый модуль показывает, как будет написан следующий модуль. Чистые программы написаны настолько хорошо, что вы этого даже не замечаете. Благодаря автору код выглядит до смешного простым, как и все действительно выдающиеся творения.

А как насчет представления Уорда о красоте? Все мы жаловались на языки, не предназначенные для решения наших задач. Однако утверждение Уорда возлагает ответственность на нас. Он говорит, что при чтении красивого кода язык кажется созданным для решения конкретной задачи! Следовательно, мы сами должны позаботиться о том, чтобы язык казался простым! Языковые фанатики, задумайтесь! Не язык делает программы простыми. Программа выглядит простой благодаря работе программиста!

Школы мысли

А как насчет меня (Дядюшка Боб)? Что я думаю по поводу чистого кода? Эта книга расскажет вам во всех подробностях, что я и мои соратники думаем о чистом коде. Вы узнаете, как, по нашему мнению, должно выглядеть чистое имя переменной, чистая функция, чистый класс и т. д. Мы излагаем свои мнения в виде беспрекословных истин и не извиняемся за свою категоричность. Для нас, на данном моменте наших карьер, они *являются* беспрекословными истинами. Они составляют нашу *школу мысли* в области чистого кода.

Мастера боевых искусств не достигли единого мнения по поводу того, какой из видов единоборств является лучшим, а какие приемы — самыми эффективными. Часто ведущие мастера создают собственную школу и набирают учеников. Так появилась школа дзю-дзюцу Грейси, основанная семьей Грейси в Бразилии. Так появилась школа дзю-дзюцу Хаккорю, основанная Окуямой Рюхо в Токио. Так появилась школа Джит Кун-до, основанная Брюсом Ли в Соединенных Штатах.



Ученики этих разных школ погружаются в учение основателя школы. Они посвящают себя изучению того, чему учит конкретный мастер, часто отказываясь от учений других мастеров. Позднее, когда уровень их мастерства возрастет, они могут стать учениками другого мастера, чтобы расширить свои познания и проверить их на практике. Некоторые переходят к совершенствованию своих навыков, открывают новые приемы и открывают собственные школы.

Ни одна из этих разных школ не обладает *абсолютной истиной*. Тем не менее в рамках конкретной школы мы действуем так, будто ее учение и арсенал приемов верны. Именно так и следует тренироваться в школе Хаккорю или Джит Кун-до. Но правильность принципов в пределах одной школы не делает ошибочными учения других школ.

Считайте, что эта книга является описанием Школы учителей Чистого кода. В ней представлены те методы и приемы, которыми мы сами пользуемся в своем искусстве. Мы утверждаем, что если вы последуете нашему учению, то это принесет вам такую же пользу, как и нам, и вы научитесь писать чистый и профессиональный код. Но не стоит думать, что наше учение «истинно» в каком-то абсолютном смысле. Существуют другие школы и мастера, которые имеют ничуть не меньше оснований претендовать на профессионализм. Не упускайте возможности учиться у них.

В самом деле, многие рекомендации в этой книге противоречивы. Вероятно, вы согласитесь не со всеми из них. Возможно, против некоторых вы будете яростно протестовать. Это нормально. Мы не претендуем на абсолютную истину. С дру-

гой стороны, приведенные в книге рекомендации являются плодами долгих, непростых размышлений. Мы пришли к ним после десятилетий практической работы, непрестанных проб и ошибок. Независимо от того, согласитесь вы с нами или нет, нашу точку зрения стоит по крайней мере узнать и уважать.

Мы — авторы

Поле @author комментария javadoc говорит о том, кто мы такие. Мы — авторы. А как известно, у каждого автора имеются свои читатели. Автор несет ответственность за то, чтобы хорошо изложить свои мысли читателям. Когда вы в следующий раз напишете строку кода, вспомните, что вы — автор, и пишете для читателей, которые будут оценивать плоды вашей работы.

Кто-то спросит: так ли уж часто читается наш код? Разве большая часть времени не уходит на его написание?

Вам когда-нибудь доводилось воспроизводить запись сеанса редактирования? В 80-х и 90-х годах существовали редакторы, записывавшие все нажатия клавиш (например, Emacs). Вы могли проработать целый час, а потом воспроизвести весь сеанс, словно ускоренное кино. Когда я это делал, результаты оказывались просто потрясающими. Большинство операций относилось к прокрутке и переходу к другим модулям!

Боб открывает модуль.

Он находит функцию, которую необходимо изменить.

Задумывается о последствиях.

Ой, теперь он переходит в начало модуля, чтобы проверить инициализацию переменной.

Снова возвращается вниз и начинает вводить код.

Стирает то, что только что ввел.

Вводит заново.

Еще раз стирает!

Вводит половину чего-то другого, но стирает и это!

Прокручивает модуль к другой функции, которая вызывает изменяемую функцию, чтобы посмотреть, как она вызывается.

Возвращается обратно и восстанавливает только что стертый код.

Задумывается.

Снова стирает!

Открывает другое окно и просматривает код subclasses. Переопределяется ли в нем эта функция?

...

В общем, вы поняли. На самом деле соотношение времени чтения и написания кода превышает 10:1. Мы постоянно читаем свой старый код, поскольку это необходимо для написания нового кода.