12+

VIBE CODING

ПОЛНОЕ РУКОВОДСТВО

Виктор Грек

Виктор Грек

Vibe Coding. Полное руководство

«Издательские решения»

Грек В.

Vibe Coding. Полное руководство / В. Грек — «Издательские решения»,

Код обесценился. Впервые в истории то, что ты хочешь создать, важнее того, умеешь ли ты программировать. Эта книга — не учебник по ИИ-инструментам и не манифест «теперь каждый может кодить». Это система мышления для тех, кто хочет создавать настоящие продукты в мире, где код генерируется за секунды, а конкурент копирует твои фичи за вечер.

Содержание

Vibe Coding. Полное руководство

Виктор Грек

ISBN 978-5-0070-0078-9
Создано в интеллектуальной издательской системе Ridero

Часть 1. Почему именно Vibe Code?

Быстрый просмотр социальных сетей, посвященных разработке программного обеспечения, покажет широкое распространение противоречивых мнений о «вайб-кодировании». Многие невероятно циничны, утверждая, что оно приводит к появлению тысяч строк неподдерживаемого и ненадежного кода, в то время как другие говорят, что оно коренным образом изменит разработку программного обеспечения.

Что такое Vibe Coding: программирование на кураже или новая реальность?

Vibe Coding (или «кодинг по вайбу») — это ироничное название нового подхода к разработке. Его суть проста: вы общаетесь с нейросетью на языке высокоуровневых идей, а не конкретных функций. Вы задаете общее направление, принимаете правки от ИИ и следите за общим «духом» проекта, не погружаясь в дебри реализации.

Видение Андрея Карпати

Андрей Карпати, один из идеологов индустрии, описывает будущее, где программист превращается в режиссера. Общение с кодом становится диалогом: «Я вижу задачу, озвучиваю её, запускаю решение — и в большинстве случаев всё просто работает». В этой парадигме намерение важнее реализации.

Реальность: ловушка «последних 30%»

Звучит заманчиво, но здесь разработчиков поджидает суровая реальность. Если первые **70%** функционала на вайбе строятся стремительно, то оставшиеся **30%** без глубокой инженерной базы могут стать непреодолимым барьером.

Вот с чем сталкиваются «вайб-кодеры»:

Эффект «Шаг вперед, два назад»: попытка исправить один баг с помощью нейросети часто порождает цепочку новых ошибок в самых неожиданных местах.

Скрытые долги: без экспертного взгляда код превращается в «черный ящик», который невозможно поддерживать или масштабировать в будущем.

Закон убывающей отдачи: инструменты ИИ дают мощный буст опытным инженерам, но могут запутать новичка, который не понимает, что именно ИИ «наворотил» под капотом.

Риски безопасности: как шутят в индустрии, *«Vibe coding — это весело и задорно ровно до тех пор, пока вы случайно не сольете ключи от базы данных в открытый доступ»*.

Vibe Coding ≠ «Тяп-ляп»

Важно понимать: «кодинг по вайбу» — это не синоним плохого кода. Это специфический метод прототипирования, когда вы доверяете ИИ настолько, что даже не читаете листинг перед запуском. Это отлично работает для проверки гипотез, но для создания серьезных систем одного «вайба» недостаточно.

Новый стандарт: Разработка с ИИ (AI-Engineering)

Будущее не за слепым копированием ответов чат-бота, а за **AI-Engineering**. Это осознанный синтез творческого драйва и строгих инженерных стандартов.

Этот подход требует:

Четких спецификаций и архитектурного планирования.

Глубокого контроля качества.

Грамотного симбиоза человека и алгоритма.

Итог: Мы переходим в эру, где продукт должен быть не просто рабочим «здесь и сейчас», но безопасным, поддерживаемым и надежным в долгой перспективе. Вайб — это искра, но инженерия — это двигатель.

Философия: Доверие к машине

Vibe-кодирование — это не просто техника. Это философский подход к взаимоотношениям между разработчиками и кодом.

Конец кода как священного текста

На протяжении десятилетий в культуре программирования исходный код рассматривался как нечто, что нужно тщательно прорабатывать, проверять, оптимизировать и понимать. Проверка кода — это ритуал. Чистый код — это моральная добродетель. Понимание каждой строки — профессиональная обязанность.

В концепции Vibe-кодирования этот подход полностью отвергается. Он рассматривает код как одноразовый посредник между намерениями человека и работающим программным обеспечением. Код не имеет значения. Важно поведение.

Это не так радикально, как кажется. Большинство специалистов в области программного обеспечения уже взаимодействуют с уровнями абстракции, которые они не до конца понимают:

Немногие веб-разработчики читают внутреннее устройство TCP-пакетов.

Немногие разработчики приложений проверяют результаты работы компилятора.

Немногие разработчики React понимают алгоритм согласования волокон.

Немногие пользователи SQL отслеживают планы выполнения запросов для каждого запроса. Vibe-кодирование просто добавляет еще один уровень: ИИ становится компилятором естественного языка. Четыре столпа

Намерение важнее реализации. Вопрос «Что это должно делать?» заменяет вопрос «Как мне это построить?».

Аргумент абстракции

Сторонники рассматривают программирование с учетом особенностей восприятия реальности как естественное развитие абстракции в программировании:

1950-е годы
Машинный код → Ассемблер
«Вам больше не нужно писать двоичные коды операций!»

1970-е годы
Сборка → C
«Вам больше не нужно вести кассы!»

1990-е годы
C → Python / Java
«Вам больше не нужно управлять памятью!»

2010-е годы
Фреймворки / Облачные технологии
«Вам больше не нужно управлять серверами!»

2025
Естественный язык → Код
«Вам больше не нужно писать код!»

На каждом этапе перехода пуристы предупреждали, что разработчики теряют важнейшие навыки. На каждом этапе перехода расширенная абстракция позволяла большему числу людей создавать больше вещей.

Однако контраргумент вполне обоснован:

Все предыдущие абстракции по-прежнему обладали детерминированным поведением. Ассемблер всегда компилируется одинаково. C всегда выделяет память одинаково. Генерация кода в ИИ носит вероятностный характер — один и тот же запрос может каждый раз выдавать разный код с разными ошибками. Это действительно новый тип уровня абстракции.

Глава 1. Новая экономика софта

Код обесценился. Что дальше?

В 2014 году час работы фронтенд-разработчика на фрилансе стоил $50—80. В 2024 — ту же работу Claude или GPT делает за 30 секунд и $0.03 токенов.

Это не снижение цен. Это обнуление. Фундаментальный экономический сдвиг, сравнимый с тем, что произошло с фотографией, когда появились смартфоны. Фотография не исчезла — но «умение нажать на кнопку» перестало быть навыком, за который платят.

С кодом произошло то же самое. Код стал коммодити — сырьевым товаром. Как электричество. Как вода из крана. Он есть везде, он дёшев, и сам по себе он больше не является конкурентным преимуществом.

Вопрос уже не «кто умеет писать код?». Вопрос — «кто понимает, что именно нужно построить?».

Инверсия ценности

Классическая цепочка создания софта выглядела так:

Идея → Дизайн → Код → Тестирование → Деплой

И ценность в этой цепочке распределялась соответствующе: больше всего денег уходило на «Код». Самые высокие зарплаты — у тех, кто писал. Самые дорогие команды — из тех, кто строил. Весь венчурный капитал в первую очередь шёл на найм инженеров.

Теперь эта пирамида перевернулась.

Код генерируется. Тестирование автоматизируется. Деплой — одна кнопка. А вот «Идея» и «Дизайн» — это то, что ИИ не может сделать за тебя. Не потому что не хватает мощностей. А потому что у ИИ нет контекста твоей жизни, твоего рынка, твоего пользователя.

Ценность сместилась с исполнения на понимание.

Это не абстрактная философия. Это новая экономическая реальность.

Раньше основатель стартапа без технического кофаундера был как режиссёр без камеры. Он мог иметь лучший сценарий в мире — и всё равно не снять фильм. Сегодня камера бесплатна, бесконечно терпелива и понимает естественный язык.

В марте 2024 года Питер Левелс (Pieter Levels) — один человек, без команды, без офиса — запускал продукты с выручкой в миллионы долларов. Его стек: ноутбук, PHP, jQuery и ИИ-ассистент.

Это не аномалия. Это новая норма.

Раньше для запуска серьёзного продукта нужно было: 3—5 инженеров, дизайнер, DevOps, QA, проджект-менеджер. Это 9 месяцев найма, $500K—1M burn rate и молитва, что рынок не изменится, пока ты нанимаешь.

Сегодня один человек с правильным пониманием проблемы и навыком «разговора с ИИ» может:

— собрать рабочий MVP за выходные, — провалидировать гипотезу на реальных пользователях в понедельник, — итерировать быстрее, чем команда из десяти человек.

Не потому что он «лучше». А потому что у него нулевая координационная нагрузка. Нет митингов. Нет согласований. Нет «давайте обсудим архитектуру в следующий четверг». Есть только: намерение → промпт → результат → обратная связь → следующая итерация.

Скорость принятия решений стала самой недооценённой суперсилой в создании софта.

И здесь возникает парадокс, который меняет всё: раньше скорость разработки была функцией размера команды. Больше людей — быстрее продукт. Теперь скорость разработки — обратная функция размера команды. Меньше людей — быстрее продукт. Потому что главный bottleneck — это не написание кода. Это коммуникация между людьми, которые его пишут.

Когда код перестаёт быть барьером, что остаётся?

Остаётся то, что всегда было за кадром: вкус, насмотренность, понимание людей.

Подумай: почему одни продукты «цепляют», а другие — нет? Технически они могут быть идентичны. Один и тот же фреймворк, одна и та же база данных, один и тот же хостинг. Но один продукт зарабатывает миллионы, а другой — пустует.

Разница не в коде. Разница в том, как продукт чувствуется.

Это не метафора. Это буквально то, как работает пользовательская экономика. Люди выбирают продукты не по количеству фич. Они выбирают по ощущению: «это понимает мою проблему» vs «это ещё один шаблонный SaaS».

И здесь гуманитарный бэкграунд перестаёт быть «слабостью» и становится суперсилой.

Человек, который читал Достоевского, понимает мотивацию пользователя глубже, чем тот, кто выучил только паттерны GoF. Человек, который изучал дизайн, видит интерфейс как

систему значений, а не набор компонентов из Figma. Человек, который разбирается в философии, может задать правильный вопрос — «зачем этот продукт существует?» — до того, как написана первая строка промпта.

Vibe coding — это не программирование для тех, кто не умеет программировать. Это программирование для тех, кто умеет думать.

Если код стал коммодити, где теперь создаётся реальная стоимость? Ответ: в трёх слоях, которые ИИ не может заменить.

Первый слой: Видение. Способность увидеть проблему, которую не видит рынок. Не «ещё один трекер задач», а «почему подрядчики в строительной отрасли теряют 30% проектов из-за плохой коммуникации». Видение — это не креативность в вакууме. Это глубокое понимание конкретной боли конкретных людей.

Второй слой: Вкус. Умение отличить «работает» от «работает правильно». ИИ сгенерирует 10 вариантов дизайна. Но выбрать тот, который резонирует с аудиторией — это решение человека. Вкус — не врождённый талант. Это натренированная насмотренность. Ты формируешь его, когда пользуешься хорошими продуктами, читаешь хорошие книги, изучаешь хорошие системы.

Третий слой: Контекст. Знание, которое невозможно загуглить. Почему именно этот сегмент рынка готов платить. Какие культурные нюансы определяют UX в конкретной стране. Как устроены отношения между участниками индустрии. Контекст — это то, что делает тебя незаменимым, потому что ИИ работает с обобщениями, а ты — с частностями.

Одно из самых радикальных последствий этого сдвига — переформатирование ролей в стартапах.

Десятилетиями существовал архетип: «визионер + технарь». Стив Джобс и Стив Возняк. Бизнес-кофаундер и технический кофаундер. Один знает «что строить», другой знает «как строить».

Этот архетип устарел. Не потому что технические навыки не нужны — они нужны. Но «как строить» больше не требует отдельного человека. Оно требует ясного промпта.

Новый архетип: один человек, который знает и «что», и «зачем», и умеет внятно объяснить это ИИ.

Это не значит, что команды исчезнут. Команды нужны для масштаба, для поддержки, для роста. Но точка входа изменилась навсегда. Раньше минимальная жизнеспособная команда — 2—3 человека. Сегодня минимальная жизнеспособная команда — 1 человек и его ясность мышления.

Мы переходим от экономики навыков к экономике намерений.

В экономике навыков ценился ответ на вопрос «что ты умеешь делать?». Я умею писать на Python. Я знаю React. Я развёртываю Kubernetes-кластеры. Навык был товаром, который ты продавал рынку.

В экономике намерений ценится ответ на другой вопрос: «что ты хочешь создать и почему?». Потому что «как» — берёт на себя машина.

Это не делает навыки бесполезными. Но это радикально меняет их иерархию:

Навык писать код → полезный, но не уникальный. Навык понимать архитектуру → ценный, потому что ИИ создаёт код, но не видит систему целиком. Навык формулировать намерение → критически важный, потому что без него ИИ — просто мощный генератор шума. Навык отличать хорошее от плохого → бесценный, потому что машина не знает, что «хорошо» — она знает только, что «статистически вероятно».

Если ты читаешь эту книгу, у тебя есть одно из двух:

Либо идея, которую ты не мог реализовать, потому что не умел кодить. Либо навык кодинга, который ты боишься потерять, потому что ИИ «забирает работу».

В обоих случаях ответ один: переоценка того, что на самом деле является ценным.

Код — не ценность. Код — это мост. Ценность — на том берегу: это проблема, которую ты решаешь, и человек, для которого ты её решаешь.

Вся остальная книга — про то, как построить этот мост быстрее, надёжнее и элегантнее, чем когда-либо было возможно.

Но сначала тебе нужно точно знать, куда он ведёт.

Единорог из одного человека

В марте 2024 года Питер Левелс (Pieter Levels) — один человек, без команды, без офиса — запускал продукты с выручкой в миллионы долларов. Его стек: ноутбук, PHP, jQuery и ИИ-ассистент.

Это не аномалия. Это новая норма.

Раньше для запуска серьёзного продукта нужно было: 3—5 инженеров, дизайнер, DevOps, QA, проджект-менеджер. Это 9 месяцев найма, $500K—1M burn rate и молитва, что рынок не изменится, пока ты нанимаешь.

Сегодня один человек с правильным пониманием проблемы и навыком «разговора с ИИ» может:

— собрать рабочий MVP за выходные, — провалидировать гипотезу на реальных пользователях в понедельник, — итерировать быстрее, чем команда из десяти человек.

Не потому что он «лучше». А потому что у него нулевая координационная нагрузка. Нет митингов. Нет согласований. Нет «давайте обсудим архитектуру в следующий четверг». Есть только: намерение → промпт → результат → обратная связь → следующая итерация.

Скорость принятия решений стала самой недооценённой суперсилой в создании софта.

И здесь возникает парадокс, который меняет всё: раньше скорость разработки была функцией размера команды. Больше людей — быстрее продукт. Теперь скорость разработки — обратная функция размера команды. Меньше людей — быстрее продукт. Потому что главный bottleneck — это не написание кода. Это коммуникация между людьми, которые его пишут.

Софт как искусство: возвращение гуманитариев

Когда код перестаёт быть барьером, что остаётся?

Остаётся то, что всегда было за кадром: вкус, насмотренность, понимание людей.

Подумай: почему одни продукты «цепляют», а другие — нет? Технически они могут быть идентичны. Один и тот же фреймворк, одна и та же база данных, один и тот же хостинг. Но один продукт зарабатывает миллионы, а другой — пустует.

Разница не в коде. Разница в том, как продукт чувствуется.

Это не метафора. Это буквально то, как работает пользовательская экономика. Люди выбирают продукты не по количеству фич. Они выбирают по ощущению: «это понимает мою проблему» vs «это ещё один шаблонный SaaS».

И здесь гуманитарный бэкграунд перестаёт быть «слабостью» и становится суперсилой.

Человек, который читал Достоевского, понимает мотивацию пользователя глубже, чем тот, кто выучил только паттерны GoF. Человек, который изучал дизайн, видит интерфейс как систему значений, а не набор компонентов из Figma. Человек, который разбирается в философии, может задать правильный вопрос — «зачем этот продукт существует?» — до того, как написана первая строка промпта.

Vibe coding — это не программирование для тех, кто не умеет программировать. Это программирование для тех, кто умеет думать.

Три слоя новой ценности

Если код стал коммодити, где теперь создаётся реальная стоимость? Ответ: в трёх слоях, которые ИИ не может заменить.

Первый слой: Видение. Способность увидеть проблему, которую не видит рынок. Не «ещё один трекер задач», а «почему подрядчики в строительной отрасли теряют 30% проектов из-за плохой коммуникации». Видение — это не креативность в вакууме. Это глубокое понимание конкретной боли конкретных людей.

Второй слой: Вкус. Умение отличить «работает» от «работает правильно». ИИ сгенерирует 10 вариантов дизайна. Но выбрать тот, который резонирует с аудиторией — это решение человека. Вкус — не врождённый талант. Это натренированная насмотренность. Ты формируешь его, когда пользуешься хорошими продуктами, читаешь хорошие книги, изучаешь хорошие системы.

Третий слой: Контекст. Знание, которое невозможно загуглить. Почему именно этот сегмент рынка готов платить. Какие культурные нюансы определяют UX в конкретной стране. Как устроены отношения между участниками индустрии. Контекст — это то, что делает тебя незаменимым, потому что ИИ работает с обобщениями, а ты — с частностями.

Смерть «технического кофаундера»

Одно из самых радикальных последствий этого сдвига — переформатирование ролей в стартапах.

Десятилетиями существовал архетип: «визионер + технарь». Стив Джобс и Стив Возняк. Бизнес-кофаундер и технический кофаундер. Один знает «что строить», другой знает «как строить».

Этот архетип устарел. Не потому что технические навыки не нужны — они нужны. Но «как строить» больше не требует отдельного человека. Оно требует ясного промпта.

Новый архетип: один человек, который знает и «что», и «зачем», и умеет внятно объяснить это ИИ.

Это не значит, что команды исчезнут. Команды нужны для масштаба, для поддержки, для роста. Но точка входа изменилась навсегда. Раньше минимальная жизнеспособная команда — 2—3 человека. Сегодня минимальная жизнеспособная команда — 1 человек и его ясность мышления.

Экономика намерений

Мы переходим от экономики навыков к экономике намерений.

В экономике навыков ценился ответ на вопрос «что ты умеешь делать?». Я умею писать на Python. Я знаю React. Я развёртываю Kubernetes-кластеры. Навык был товаром, который ты продавал рынку.

В экономике намерений ценится ответ на другой вопрос: «что ты хочешь создать и почему?». Потому что «как» — берёт на себя машина.

Это не делает навыки бесполезными. Но это радикально меняет их иерархию:

Навык писать код → полезный, но не уникальный. Навык понимать архитектуру → ценный, потому что ИИ создаёт код, но не видит систему целиком. Навык формулировать намерение → критически важный, потому что без него ИИ — просто мощный генератор шума. Навык отличать хорошее от плохого → бесценный, потому что машина не знает, что «хорошо» — она знает только, что «статистически вероятно».

Что это значит для тебя

Если ты читаешь эту книгу, у тебя есть одно из двух:

Либо идея, которую ты не мог реализовать, потому что не умел кодить. Либо навык кодинга, который ты боишься потерять, потому что ИИ «забирает работу».

В обоих случаях ответ один: переоценка того, что на самом деле является ценным.

Код — не ценность. Код — это мост. Ценность — на том берегу: это проблема, которую ты решаешь, и человек, для которого ты её решаешь.

Вся остальная книга — про то, как построить этот мост быстрее, надёжнее и элегантнее, чем когда-либо было возможно.

Но сначала тебе нужно точно знать, куда он ведёт.

Глава 2. Сначала Инженерия, потом Вайбы

Ловушка лёгкости

Ты прочитал предыдущую главу и, возможно, почувствовал эйфорию. Код обесценился! Один человек может всё! Гуманитарии — новые рок-звёзды!

Остановись.

Потому что именно здесь погибают 90% проектов, созданных через vibe coding. Не на старте. Не при генерации первого экрана. Они погибают на третьей неделе, когда код, который «работал», начинает работать неправильно. Когда одно изменение ломает три других. Когда ты просишь ИИ починить баг — и он чинит, одновременно создавая два новых. Когда проект превращается в то, что инженеры называют big ball of mud — ком грязи, в котором всё связано со всем и ничего нельзя тронуть.

Это не вина ИИ. Это вина отсутствия инженерной дисциплины.

Vibe coding без инженерного мышления — это как управлять Формулой-1, не зная, где тормоз. Первые два поворота — восторг. Третий — стена.

Почему классика актуальна именно сейчас

Есть книга, написанная в 1999 году. Она называется «The Pragmatic Programmer». Её авторы — Эндрю Хант и Дэвид Томас — сформулировали принципы, которые большинство программистов заучивают и потом забывают. Модульность. Ортогональность. Don't Repeat Yourself. Принцип наименьшего удивления.

Двадцать пять лет эти принципы считались «теорией для зануд». Настоящие хакеры пишут код, а не читают книжки про архитектуру.

Но вот что изменилось: когда код писал человек, он мог компенсировать плохую архитектуру интуицией. Он помнил, почему здесь стоит костыль. Он знал, что эту функцию нельзя трогать, потому что она связана с тремя другими. У него была ментальная карта проекта, пусть кривая и неполная.

ИИ не помнит ничего.

Каждый раз, когда ты открываешь новый диалог с ИИ-ассистентом, он видит код как будто впервые. У него нет истории решений. Нет понимания, почему именно так. Нет «ментальной карты». Есть только текст, который он видит прямо сейчас.

И если этот текст — хаос, ИИ произведёт ещё больший хаос. С абсолютной уверенностью и безупречным синтаксисом.

Именно поэтому принципы из «Прагматичного программиста» — не устарели. Они стали критически важными. Не для того, чтобы ты писал код руками. А для того, чтобы ты мог оценить код, который пишет машина.

DRY — или как не утонуть в копиях

DRY — Don’t Repeat Yourself — самый простой и самый нарушаемый принцип в мире vibe coding.

Вот что происходит. Ты просишь ИИ: «Сделай страницу регистрации». Он делает. Работает. Ты просишь: «Теперь сделай страницу входа». Он делает. Тоже работает. Ты просишь: «Добавь восстановление пароля». Сделано.

Три страницы. Три формы. И в каждой — своя копия валидации email. Своя копия обработки ошибок. Своя копия стилей кнопок. Не потому что ИИ глуп. А потому что ИИ решает каждую задачу изолированно, если ты не скажешь ему иначе.

Через месяц ты меняешь дизайн кнопок. Меняешь в одном месте. Забываешь про два других. Пользователь видит три разных кнопки на трёх разных страницах. Продукт выглядит так, будто его делали три разных человека в трёх разных настроениях.

Принцип DRY в эпоху vibe coding означает: прежде чем просить ИИ создать что-то новое, спроси себя — нет ли уже чего-то похожего в проекте? И если есть — скажи ИИ использовать существующее, а не генерировать с нуля.

Это звучит просто. На практике это требует дисциплины, которой у большинства «вайбкодеров» нет, потому что генерировать новое — приятно, а поддерживать существующее — скучно.

Модульность: стены, которые спасают

Модульность — это когда систему можно разделить на независимые части, каждая из которых делает одну вещь и делает её хорошо. Если одна часть сломается — остальные продолжают работать.

В обычной разработке модульность — это хорошая практика. В vibe coding — это вопрос выживания.

Почему? Потому что ИИ работает с контекстом. Чем больше кода ты скармливаешь ему в одном запросе, тем менее точным становится результат. Монолитный файл на 3000 строк — это кошмар для ИИ. Он теряет нить. Путает функции. Вносит изменения не туда.

Модульный проект — 20 файлов по 150 строк — это мечта. ИИ видит маленький, понятный модуль. Меняет только его. Не трогает остальное. Результат предсказуем.

Модульность в vibe coding — это не про «красивый код». Это про управляемость. Ты создаёшь стены между частями системы, чтобы ИИ не мог случайно снести несущую конструкцию, ремонтируя ванную.

Практическое правило: если файл больше 200 строк — разбей. Если функция делает больше одной вещи — раздели. Если модуль знает слишком много о другом модуле — изолируй. Не потому что так написано в учебнике. А потому что иначе ИИ превратит твой проект в карточный домик.

Роль Главного Редактора

Здесь нужно зафиксировать сдвиг мышления, который определяет всё остальное в этой книге.

Ты больше не программист. Ты — главный редактор.

Представь редактора газеты. Он не пишет каждую статью сам. У него журналисты — быстрые, продуктивные, иногда блестящие. Но редактор не публикует то, что ему приносят, без проверки. Он знает: журналист может ошибиться в фактах, притянуть вывод, пропустить контекст. Задача редактора — поймать это до того, как увидит читатель.

ИИ — это твой журналист. Невероятно быстрый, очень начитанный, но без собственного суждения. Он напишет тебе всё, что попросишь. Но он не знает, правильно ли это. Не знает, подходит ли это архитектуре. Не знает, не противоречит ли это тому, что он написал вчера.

Твоя работа — знать.

Это значит: каждый кусок кода, который генерирует ИИ, проходит через твой «редакторский фильтр». Не построчно — ты не обязан понимать каждую точку с запятой. Но структурно:

— Этот модуль делает одну вещь или десять? — Он дублирует что-то, что уже есть? — Он использует данные, к которым не должен иметь доступа? — Если я уберу его, сломается ли что-то ещё?

Это и есть «аудит ИИ-кода» — и это навык, который отделяет тех, кто строит реальные продукты, от тех, кто строит демо для Twitter.

Внутренняя библиотека суждений

Хороший редактор отличается от плохого одним: у него есть внутренняя библиотека суждений. Он читал достаточно текстов, чтобы мгновенно чувствовать — «это хорошо» или «что-то не так».

У инженера то же самое. Он видел достаточно систем, чтобы отличить хорошую архитектуру от «цифровых декораций» — кода, который выглядит красиво, но разваливается при первом столкновении с реальностью.

Как строить эту библиотеку, если ты не писал код десять лет?

Читай чужой код. Не для того, чтобы копировать — для того, чтобы видеть паттерны. Открой любой популярный open-source проект на GitHub. Посмотри на структуру. Как разделены файлы? Как названы функции? Где лежат тесты? Ты не обязан всё понимать. Ты тренируешь глаз.

Ломай свои проекты нарочно. Возьми рабочий проект и удали один файл. Что сломалось? Всё — значит, архитектура хрупкая. Только один экран — значит, модульность работает.

Проси ИИ объяснить. Это самый недооценённый приём: вместо «напиши мне код» скажи «объясни мне этот код как архитектор». ИИ расскажет, где связи сильные, где слабые, где потенциальные точки отказа. Ты не будешь писать — но ты будешь понимать.

Со временем эта библиотека суждений становится твоим главным активом. Не промпты. Не шаблоны. Не «секретные техники». А натренированное чутьё на то, что работает и что рухнет.

Инженерия как форма уважения

Последнее, что стоит сказать об инженерии, и это не про код.

Писать чистый, модульный, понятный код — это форма уважения. К себе через полгода, когда ты вернёшься к проекту и не вспомнишь, зачем здесь стоит эта функция. К другим людям, которые будут работать с твоей системой. К пользователям, которые доверяют тебе свои данные, своё время, своё внимание.

Vibe coding снял барьер входа. Это великая вещь. Но снятый барьер входа — не отмена ответственности. Напротив: когда создавать софт может каждый, качество становится тем, что отличает серьёзный продукт от игрушки.

Инженерная дисциплина — это не противоположность вайбам. Это фундамент, на котором вайбы работают.

Глава 3. Роль Главного Редактора

Есть старая шутка про программистов: их работа — превращать кофе в код. Два монитора, механическая клавиатура, 14 часов печатания, стек технологий в резюме длиной в два экрана.

Эта эпоха закончилась. Не шутка — реальность.

Человек, который сегодня создаёт софт через ИИ, не печатает код. Он делает нечто принципиально иное: формулирует намерение, оценивает результат, корректирует направление. Снова и снова, с каждой итерацией сужая зазор между тем, что он хочет, и тем, что получает.

Это не программирование. Это редактура.

И разница между плохим и хорошим редактором — та же, что между человеком, который принимает первый черновик ИИ, и человеком, который доводит результат до точности.

Прежде чем говорить о том, как редактировать хорошо, разберём, как это делают плохо.

Типичный промпт новичка: «Сделай мне приложение для трекинга привычек».

ИИ послушно создаёт что-то. Экран. Кнопки. Какую-то логику. Это даже работает. Новичок восхищён.

Проблема в том, что ИИ принял за тебя 200 решений, о которых ты даже не подумал. Какая структура данных? Он выбрал. Где хранить? Он решил. Как выглядит «привычка» — с повторением, без, с напоминаниями, с аналитикой? Он додумал. Какой фреймворк? Он взял привычный.

Ты получил не свой продукт. Ты получил статистически среднее представление ИИ о том, что такое «трекер привычек». Это как попросить редактора журнала: «напиши статью про что-нибудь интересное». Он напишет. Но это будет его статья, не твоя.

Хороший промпт — это не описание результата. Это передача контекста и ограничений.

Управление ИИ как редактор работает на трёх уровнях. Каждый следующий — глубже, точнее и даёт лучший результат.

Уровень 1: Что. Большинство людей застревают здесь. «Сделай форму входа». «Добавь корзину». «Создай дашборд». Это уровень заказа в ресторане. Ты получишь что-то съедобное, но шеф-повар решает всё остальное.

Уровень 2: Что + Почему. «Сделай форму входа. Наша аудитория — люди старше 50, которые не доверяют технологиям. Поэтому форма должна быть максимально простой, с крупными элементами и без отвлекающих элементов». Теперь ИИ не просто выполняет задачу — он принимает решения в заданном направлении. Размер шрифта. Количество полей. Тон текста на кнопке. Всё выстраивается вокруг «почему».

Уровень 3: Что + Почему + Как не надо. «Сделай форму входа для людей старше 50. Не используй плейсхолдеры вместо лейблов — они исчезают и путают. Не делай пароль скрытым по умолчанию. Не добавляй OAuth-кнопки — „войти через Google“ пугает эту аудиторию». Это уровень, на котором ты закрываешь пространство для ошибок. Ты говоришь ИИ не только куда идти, но и куда не надо.

Третий уровень кажется избыточным. Но именно он отличает промпт, который приводит к рабочему продукту, от промпта, который приводит к бесконечным итерациям «нет, не так, переделай».

Конец ознакомительного фрагмента.