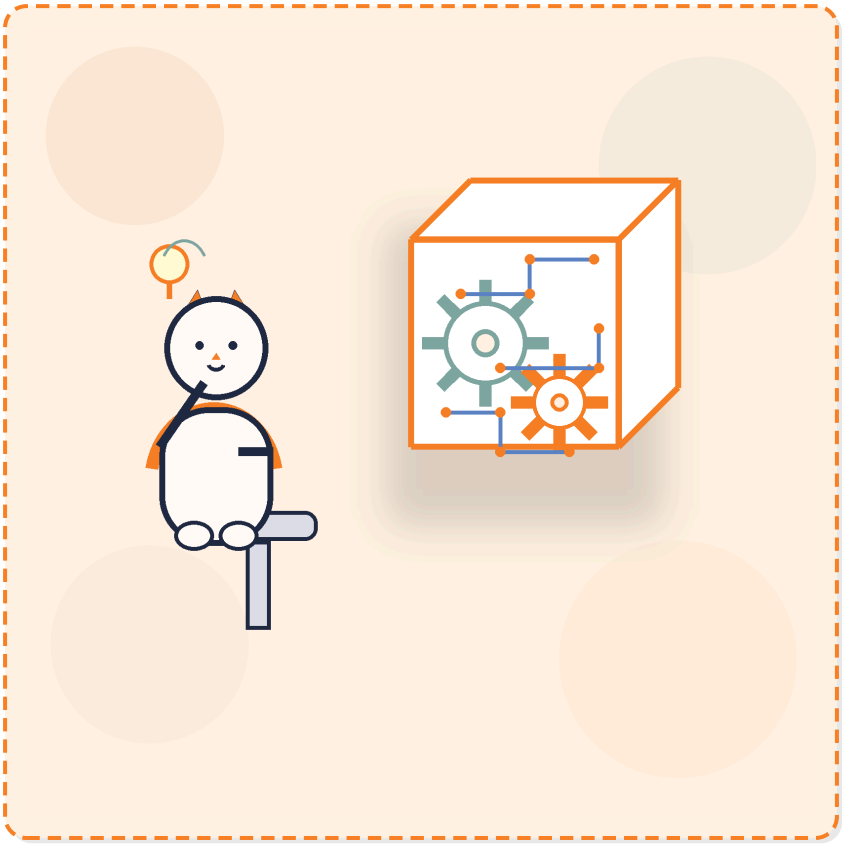
Лекция

Программирование в эпоху серых ящиков

Почему AI не ускорил программирование,
а сменил эпоху — и что с этим делать

автор **Георгий** · telegram **@OkjaGG**



Содержание

Вступление

Не ускорение, а смена эпохи

За последние два года мы все стали немного быстрее. Гораздо меньше людей заметили, что мы стали делать не то же самое — быстрее, а нечто совсем другое. **Разница между этими двумя утверждениями — это разница между эволюцией инструмента и сменой эпохи.**

Почти всё, что сегодня говорят про искусственный интеллект в разработке, крутится вокруг скорости. Быстрее писать, быстрее проверять, быстрее выкатывать. Всё это правда — и всё это мимо главного.

Главный сдвиг лежит не в скорости, а в структуре. **Код стал дешёвым.** Настолько дешёвым, что его перестало иметь смысл считать штуками. Но вместе с ценой исчезла и определённость: откуда этот кусок, насколько ему можно верить, что у него внутри. Мы больше не строим из прозрачных деталей. Мы строим из деталей полупрозрачных. И это меняет профессию сильнее, чем любое ускорение.

Сказать «AI пишет код» — это как сказать, что печатный станок ускорил переписывание книг. Формально — да. По сути — нет: **станок не ускорил переписчиков, он сделал переписчиков ненужными**, а на их месте появились издатели, редакторы и читающая публика. Сдвиг такого же масштаба происходит сейчас с нами.

01

Генерация стала дешёвой. Верификация — дорогой

Когда что-то дешевеет, первый инстинкт — радоваться. Но у технологических революций свои повадки: дешевея в одном месте, они тут же делают дорогим что-то другое. **Дефицит не исчезает. Он переезжает.**

Написать модуль теперь можно за час. Вариантов этого модуля можно сгенерировать десять, сто, тысячу. Узкое место появилось там, где его никогда не было: не в производстве, а в проверке. Действительно ли этот код работает — или он только выглядит убедительно? Он надёжен — или просто удачно прошёл пару тестовых входов? Не развалится ли на краях? Не съест ли экономику продукта? Нет ли в нём тихой хрупкости, которая проявится через полгода, в самый неудобный момент?

Это неприятная мысль для многих — она убирает магию. Выясняется, что проблема не в том, что модель плохо пишет. **Проблема в том, что система рождает вариантов больше, чем человек способен осмысленно проверить.**

И здесь происходит тихий, но фундаментальный сдвиг роли. **Инженер перестаёт быть автором — и становится судьёй.** Это разные профессии. У прокурора и судьи разная этика, разная оптика, разная

ответственность. Судья, который берёт молоток и сам чинит машину подсудимого, — это не судья. Но и инженер, который до сих пор хочет написать каждую строчку сам, — это уже не та роль, за которую ему будут платить через три года.

Главный вопрос сместился. Не «как это написать». А — **«можно ли этому верить, и на каком основании».**

02

Код — это не актив. Код — это долг

У этой мысли плохая репутация — в неё тяжело поверить, потому что она рушит интуицию, с которой выросла вся индустрия. Но попробуйте посмотреть на свою кодовую базу глазами бухгалтера, а не инженера.

Каждая строчка кода кем-то поддерживается. Каждую строчку кто-то читает, обновляет, объясняет новому сотруднику, мигрирует при смене библиотеки, исправляет при смене фреймворка. **Строчка кода — это не актив. Это микроскопический контракт о техническом обслуживании, который вы подписали с будущим.**

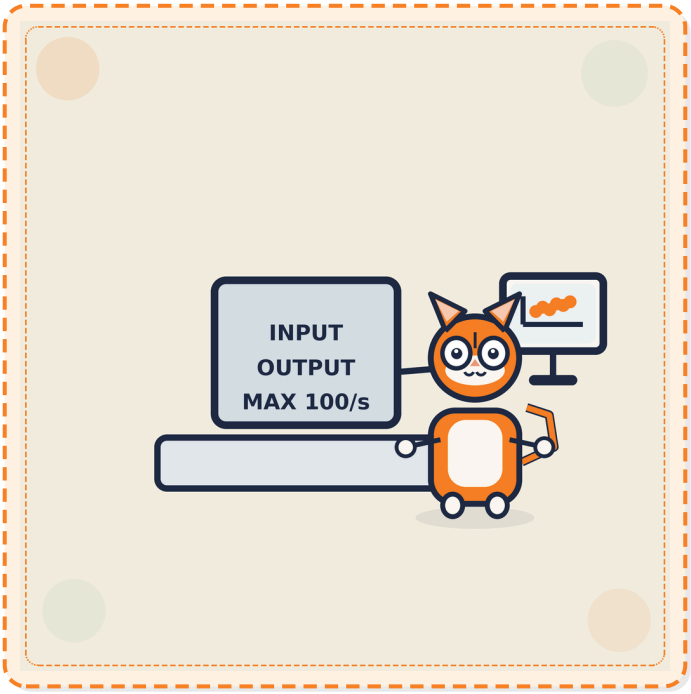
Пока код стоил дорого в производстве, его количество было честным показателем усилий, знаний, мастерства. Объём кода коррелировал с объёмом работы. Эта корреляция сломалась. Генератор за пять минут выдаёт столько кода, сколько раньше человек писал за неделю. Но обслуживать этот код всё равно придётся людям. **И количество строчек, которое один человек способен держать в голове, не выросло ни на процент.**

Зрелая компания сегодня смотрит на код не как на капитал, а как на грязную посуду после ужина. Чем её меньше — тем лучше. **Побеждает не тот, у кого кода больше. Побеждает тот, кто делает больше из**

меньшего. А ещё лучше — тот, кто вообще не пишет то, что можно не писать.

03

Мы живём в мире серых ящиков



Есть снимки, на которых видно всё. Есть снимки, на которых не видно ничего. **Большинство наших систем теперь — это третья категория: снимки, на которых видно достаточно, чтобы работать, и недостаточно, чтобы успокоиться.**

Серый ящик — это не чёрный ящик, где ты не понимаешь вообще ничего. И не белый ящик, где тебе прозрачно всё, вплоть до последнего регистра. Серый ящик — это когда ты знаешь назначение модуля, его интерфейс, его ограничения, его поведение на типовых входах, его режимы отказа, его стоимость исполнения и области доверия — **но не обязан знать его внутреннюю логику до последнего винтика.**

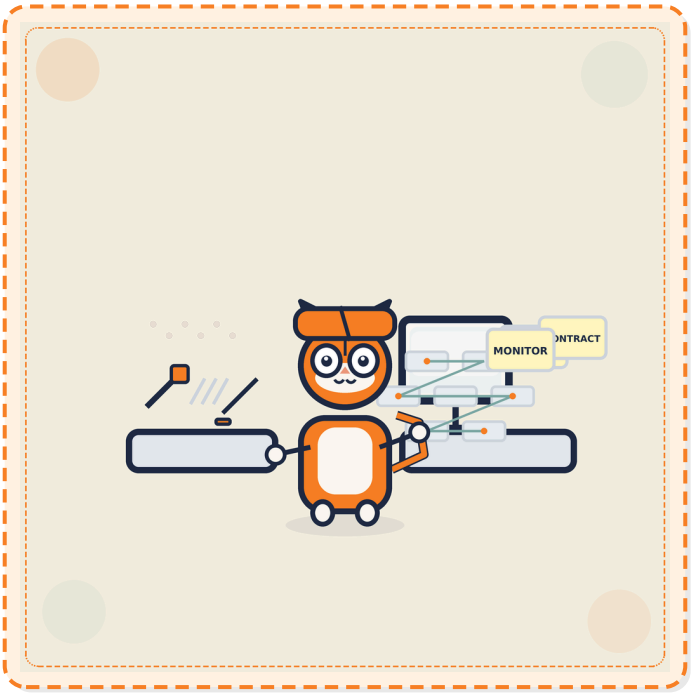
Иногда этого знания достигнуть невыгодно. Иногда невозможно. Иногда время, которое ушло бы на внутренний аудит, даст меньше пользы, чем правильно спроектированный внешний контур контроля.

Хирург, ставящий искусственный клапан сердца, не обязан знать, из какого полимера клапан сделан и как этот полимер ведёт себя на молекулярном уровне. Он обязан знать другое: при каком давлении клапан откажет, какие симптомы предвещают сбой, что делать, если клапан начнёт течь. **Это и есть зрелый серый ящик.**

Старая инженерная гордость звучала так: «Я понимаю, как это устроено внутри». Новая гордость звучит глубже: **«Я понимаю, где этой штуке можно доверять, а где нельзя. Я знаю, как она наблюдается, как ограничивается, как откатывается, и как локализуется ущерб, если она сломается».** Это не меньшая инженерия. Это другая инженерия.

04

Инженерия не умерла. Она переехала



У каждой технологической революции есть опасный момент — соблазн решить, что прежняя дисциплина больше не нужна. Обычно через пять лет выясняется, что дисциплина нужна, просто она теперь живёт по другому адресу.

Услышав всё это, легко сделать примитивный вывод: ну, значит, инженерия закончилась, теперь всё можно слепить на коленке. Нет. **Инженерия не закончилась. Она переехала.**

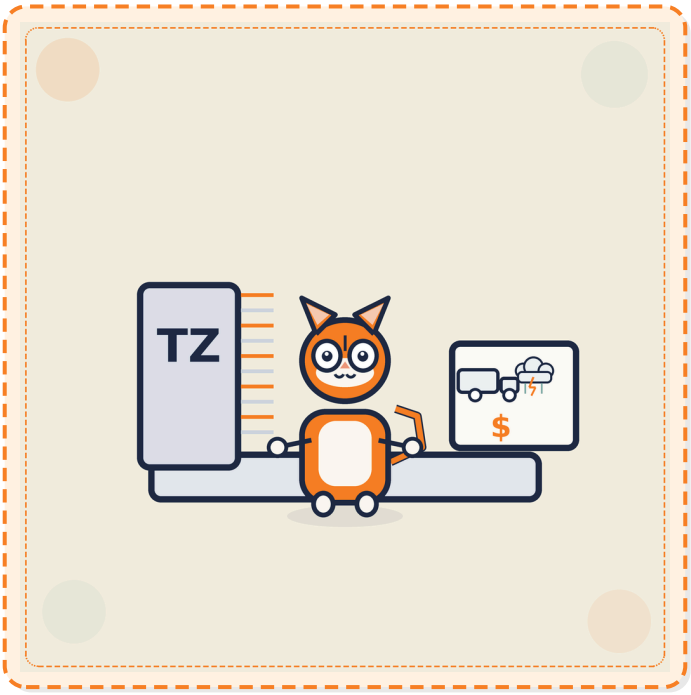
Из ручного изготовления каждой внутренней детали — в проектирование ограничений, контрактов, наблюдаемости, маршрутизации, стоимости и безопасной деградации. Сильный инженер сегодня — это не обязательно тот, кто написал каждую строчку сам. Это тот, кто умеет ответить на другие вопросы. Где в системе нужен дорогой reasoning, а где хватит дешёвого классификатора? Где можно ставить эвристику, а где нужен строгий контроль? Где нужен fallback, где — human-in-the-loop? Где latency важнее качества, а где цена ошибки выше цены inference? Где нужна свобода, а где жёсткий инвариант?

Сильный инженер проектирует вычислительную экономику системы. Один и тот же продукт можно собрать так, что операция будет

стоить 0,1 цента. А можно — так, что она будет стоить 8 центов. Пользователь внешне увидит почти одно и то же. Бизнес-модель — нет. Между этими двумя вариантами и лежит настоящее преимущество.

05

Прощай, ТЗ. Здравствуй, контекст



Теперь поднимемся на этаж выше — к тому, кто ставит задачу. Тут происходит не меньший сдвиг.

Существует старая школьная истина: чем подробнее ТЗ, тем лучше постановщик. Эта истина работала в мире, где исполнитель был медленнее постановщика и где проектирование реализации действительно требовало детализации. В мире серых ящиков она превращается в ловушку.

Чем точнее вы расписываете реализацию, тем выше вероятность, что вы закрепили не суть задачи, а случайные детали. Вы зафиксировали псевдоточность вместо точности. Хороший постановщик научился передавать другое: смысл, цель, ограничения, приоритеты, критерии успеха, недопустимые отклонения — и, что важнее всего, **степень свободы исполнителя.**

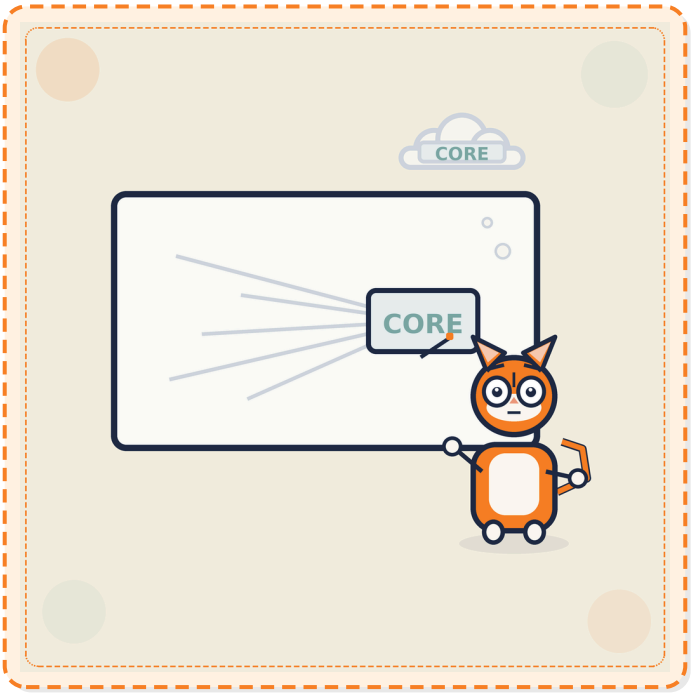
Слабый постановщик заваливает задачу подробностями. Сильный — разделяет: что обязательно, что желательно, что можно менять, что недопустимо.

Мне нужен мост через реку. Стальной или бетонный — на ваше усмотрение. Одна опора или три — ваше инженерное дело. Что я держу жёстко: мост выдерживает гружёный грузовик, не обрушивается в шторм, вписывается в береговую линию, укладывается в бюджет. Плохой заказчик моста приходит с готовым чертежом и диктует толщину каждой балки. Хороший — приносит нагрузку, климат и бюджет. **Плохой ТЗ-шник прописывает шаги. Хороший — прописывает инварианты.**

Плохая постановка — это туман или псевдоточность. **Хорошая постановка — это сжатие смысла с сохранением инвариантов.**

06

Контекст — это не поэзия. Это технология сжатия



Услышав слово «контекст», многие решают, что речь о чём-то мягком, расплывчатом, почти художественном. Ложное ощущение. **Контекст в новой разработке — это технология сжатия, требующая не меньшей дисциплины, чем схема базы данных.**

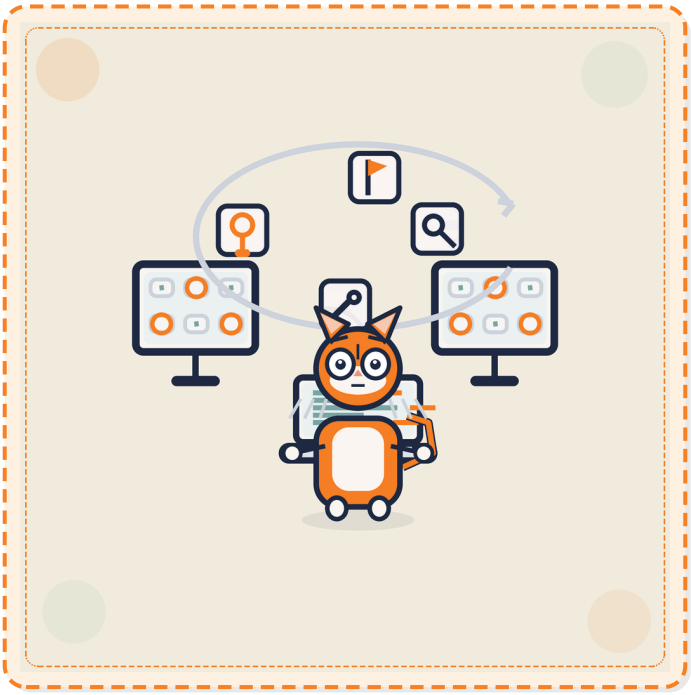
У вас в голове сложная идея. Целиком вы её не передадите — ни другому человеку, ни модели, никакими средствами. Значит, её нужно сжать. Сжать так, чтобы при потере деталей сохранилось главное. **Чтобы разные исполнители могли разойтись в реализации, но не могли разойтись в замысле.**

Это и есть проверка зрелой постановки. Дайте один и тот же контекст одному инженеру, другому инженеру, одной модели, другой модели. Если все четверо поняли задачу одинаково — в главном, — вы попали в её суть. Если поняли по-разному — вы передали не смысл, а шум.

Этот критерий намного сильнее, чем «насколько детально ты расписал шаги». **Детализация обычно мёртвая. Сжатие — живое.**

07

Главная метрика эпохи: скорость замыкания петли



Если мы живём в мире серых ящиков и дешёвой генерации, то какая метрика становится главной?

Не скорость написания кода. Не количество задач в Jira. Не velocity ради velocity. **Главный параметр — это скорость, с которой замыкается петля: идея → реализация → проверка → сигнал → корректировка.**

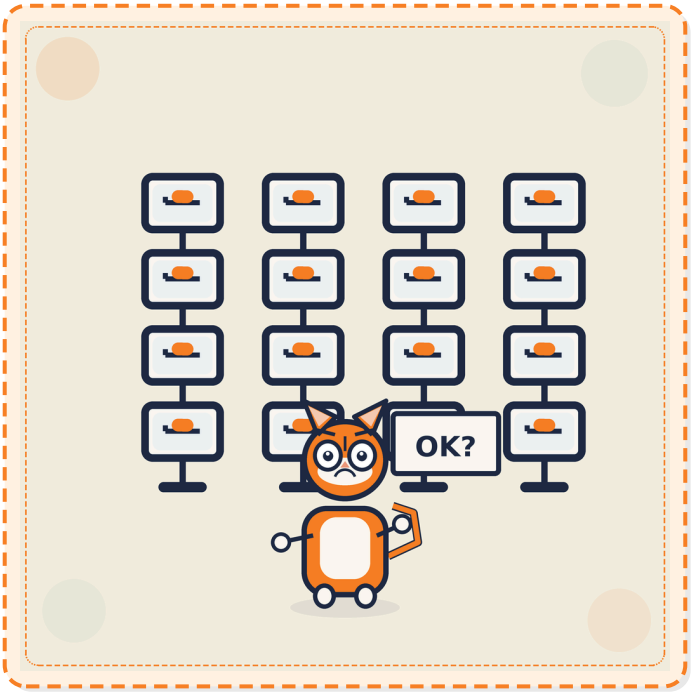
Если инженер приносит осмысленный результат раз в день — это уже очень хорошо. Раз в четыре часа — ещё лучше. Не потому, что все должны бегать в панике, а потому, что ложные траектории нужно убивать быстро. **Побеждает часто не тот, кто с первого раза придумал идеальный план. Побеждает тот, кто быстрее всех получил качественный сигнал о том, что первый план был неправильным.**

Здесь нужна жёсткая оговорка. Скорость сама по себе ничего не значит. Значение имеет только скорость *хорошего* эксперимента. Можно десять раз в день делать бессмысленные итерации и просто производить шум. Настоящая метрика — не «как быстро мы что-то собрали», а «как быстро мы получили надёжный ответ на важный вопрос».

И ещё одна вещь — неприятная, но её нужно сказать прямо. **Дешёвый инструмент требует более дорогой дисциплины мышления.** Когда прототип стоил три месяца, сама его цена заставляла думать до того, как начнёшь писать. Прототип за полдня этой защиты лишён. Вся нагрузка «думать заранее» перекладывается с экономики на характер. Плохая новость: **характер нельзя купить на Hugging Face.** Если вы думаете, что AI освободит вас от необходимости быть взрослым инженером — он сделает ровно обратное.

08

Метрика может стать театром



У серого ящика есть обратная сторона. Раз внутренности не видны, инженер всё меньше объясняет логику — и всё больше компенсирует это метриками. Это логично. Это же и опасно.

Открываешь дашборд. Сорок семь графиков. Зелёные, жёлтые, красные. Одиннадцать уровней алёртов. Полтора гигабайта метрик в день. **Простой вопрос: этой штукой сейчас можно безопасно пользоваться — или нет? Ответа нет.**

Это и есть метрический театр. **Двести метрик и ни одного решения — не приборная панель самолёта, а витрина магазина.** Количество измерений не заменяет ясность.

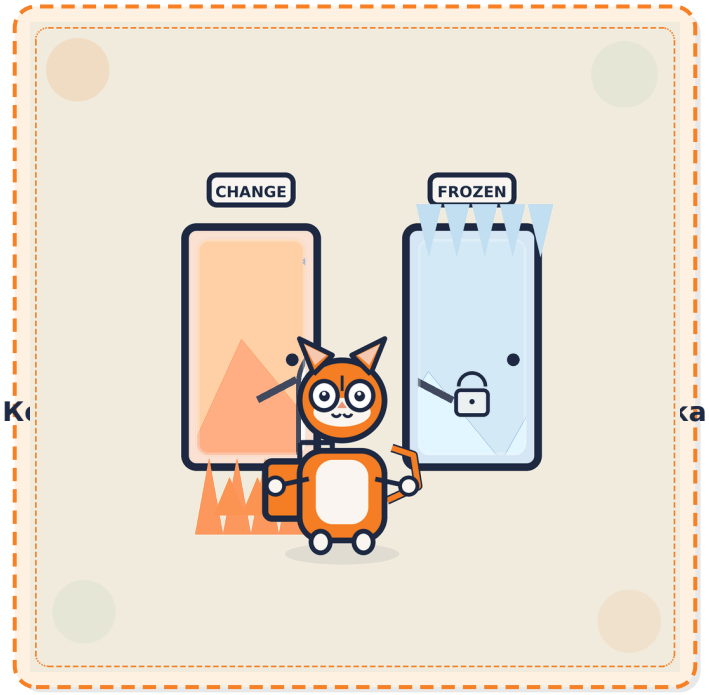
Здоровая система различает три уровня измерений. Метрики результата — делает ли модуль свою бизнес-функцию. Guardrail-метрики — не достигается ли результат опасным способом. Диагностические метрики — они нужны инженеру, когда проблема уже началась. **Управлять компанией по диагностике — ошибка.** Это всё равно что водить машину по показаниям температуры двигателя, не видя дороги.

У каждого модуля должен быть не набор графиков, а паспорт доверия. Что он должен делать. В каких условиях. Что считается нормой. Как видна деградация. Что делаем при сбое. Когда подключаем человека.

Где у нас доказательство, а где только эвристика. Паспорт доверия — это, по сути, та же идея, что советская номерная пломба на приборе: её мало, но она работает, потому что она осмысленна.

09

У системы есть горячие и замороженные регионы



У стартапов есть старое слово — вираж. Существенное изменение траектории: смена продукта, технологии, клиента, модели монетизации, архитектуры. Сейчас рядом появилось новое слово — скорость виражей. И вот здесь нужно быть очень аккуратным.

Почему виражи стали актуальнее? Потому что среда меняется быстрее самой разработки. Вы полгода делаете что-то — и выходит внешний игрок, который закрывает ваш нижний слой. Perplexity выпускает новый блок. Очередной релиз большой модели забирает функционал, который вы считали дифференциатором. Вчерашняя фича становится пунктом в чужом API.

Слабая команда в этой ситуации защищает sunk cost. Говорит: «Мы же уже начали это писать». Сильная команда говорит иначе: «Если это больше не наш moat — зачем мы продолжаем тратить на это жизнь? Берём готовый внешний блок и поднимаемся выше». **Защищать нужно не уже написанное. Защищать нужно только то, что действительно является ядром преимущества.**

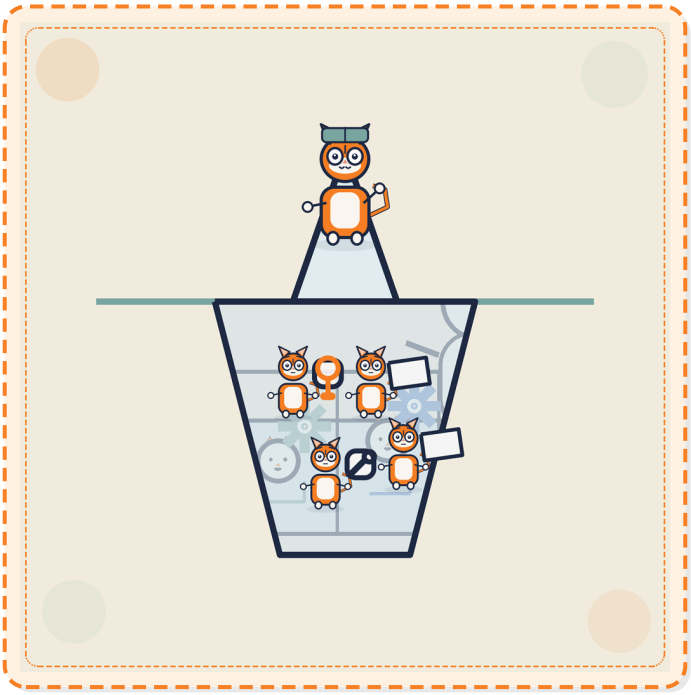
Но здесь спрятана опасная ловушка — и её пропускают чаще всего. **Не все части системы одинаково пригодны к виражам.**

В геноме есть участки, которые мутируют свободно, и есть участки, где одна-единственная замена букв означает гибель организма. Эволюция давно определила, что можно трогать, а что — нет. AI-система устроена по тому же принципу. **Промпты, роутинг, UX, порядок шагов, внутренняя маршрутизация — горячие регионы. Здесь мутации полезны, тут и должна жить скорость. Схема данных, обещание клиенту, security model, ценовая модель, контрактные ожидания рынка — замороженные регионы. Здесь каждая мутация — это не вираж, а клиническая смерть компании.**

Зрелый инженер не тот, кто смело меняет всё. Зрелый инженер точно знает, в каком регионе системы он сейчас находится. **Часто поворачивать можно только там, где откат дёшев. Виражи без страховки — не смелость. Гонка к обрыву.**

10

AI-компания: внутри жидкая, снаружи твёрдая



Из предыдущего раздела вытекает одна из главных организационных мыслей.

У хорошего корабля есть часть, на которую обрушиваются волны, и есть часть, куда волны не должны доходить никогда. У современной AI-компании структура ровно такая же — только вместо палубы и трюма у неё два слоя: быстрый и медленный.

Быстрый слой — это эксперименты, промпты, UX, workflow, маршрутизация, сборка новых комбинаций, локальные виражи. Здесь норма — делать, смотреть, переделывать, иногда выбрасывать. Медленный слой — это данные, права доступа, биллинг, обещания пользователю, безопасность, ключевые сущности, бренд-доверие, контракт с реальностью. Здесь норма — тишина, стабильность, редкие и осторожные изменения.

Внутренняя часть такой компании должна быть крайне поворотливой. Снаружи продукт должен ощущаться стабильным, предсказуемым, скучным в хорошем смысле слова.

Формула коротко: **внутри — жидкая, снаружи — твёрдая.** Если у вас внутри всё жёстко, а снаружи всё течёт — вы обещаете пользователю то, чего не удерживаете. Если у вас и внутри, и снаружи твёрдо — вы окостенели и не выживете полтора года. Если у вас и внутри, и снаружи жидко — у вас не компания, а болото.

11

Главный налог будущего — энтропия

Энтропия — не событие. Это погода. Она не случается вдруг — она просто есть, каждый день, и либо вы каждый день её убираете, либо через полгода никто уже не помнит, зачем вы начинали.

AI удешевляет не только создание решений. AI удешевляет создание хаоса. У вас появляется слишком много веток. Слишком много полуготовых модулей. Слишком много «почти одинаковых» идей, которые никто не доводит до конца, потому что сгенерировать новую проще, чем завершить старую. Слишком много локальных оптимизаций, которые оптимизируют разные куски системы в противоположных направлениях. Слишком много контекста, который уже никто целиком не держит в голове.

Возникает новый налог. Не налог на код. Не налог на инфраструктуру. **Налог на энтропию.**

Быстрые AI-команды часто проигрывают не потому, что им не хватает скорости. Они проигрывают потому, что у них распадается связность. Никто уже не помнит, почему приняли прошлое решение. Никто не знает, чем текущая версия отличается от предыдущей. Никто не вспомнит, какая гипотеза уже проверялась и с каким результатом. Где временное

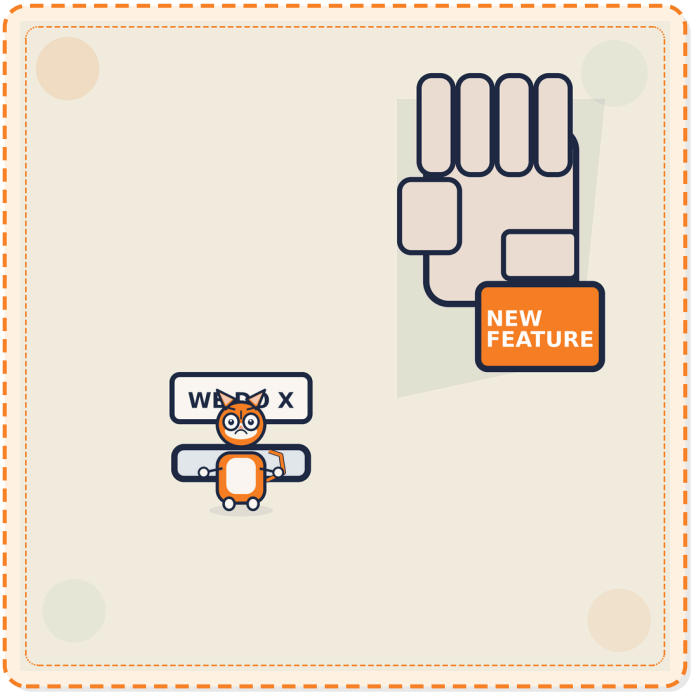
решение, а где постоянное. Какой термин что значит. Что у нас hard constraint, а что было просто пожеланием на летучке в прошлый вторник.

В AI-компании резко дорожает то, что раньше казалось бюрократической роскошью: единый словарь. Журналы решений. Версионирование не только кода, но и гипотез. Канон терминов. Кладбище идей, чтобы не тестировать одно и то же по второму кругу. Регрессии не только для кода, но и для смыслов.

Побеждает не просто быстрый. Побеждает тот, кто снижает энтропию быстрее, чем её производит.

12

Защитные рвы переезжают



Средневековые крепости строились не в чистом поле. Их ставили на самом уязвимом участке дороги — там, где враг вынужден был проходить мимо и не мог обойти. Крепость в чистом поле — это не moat. Это декорация.

С IT-компаниями происходит то же самое. Если код стремительно коммодитизируется — где строить крепость?

Не там, где её привыкли искать. Не в том, что «мы умеем написать такой же модуль» — завтра это может стать фичей у платформы на сотни миллиардов долларов. **Moat переезжают выше и глубже одновременно.**

Выше — в дистрибуцию, в доступ к пользователю, в workflow, в который вы встроены, в упаковку, в вертикальный продукт, в доверие бренда, в удобство для узкой категории клиентов.

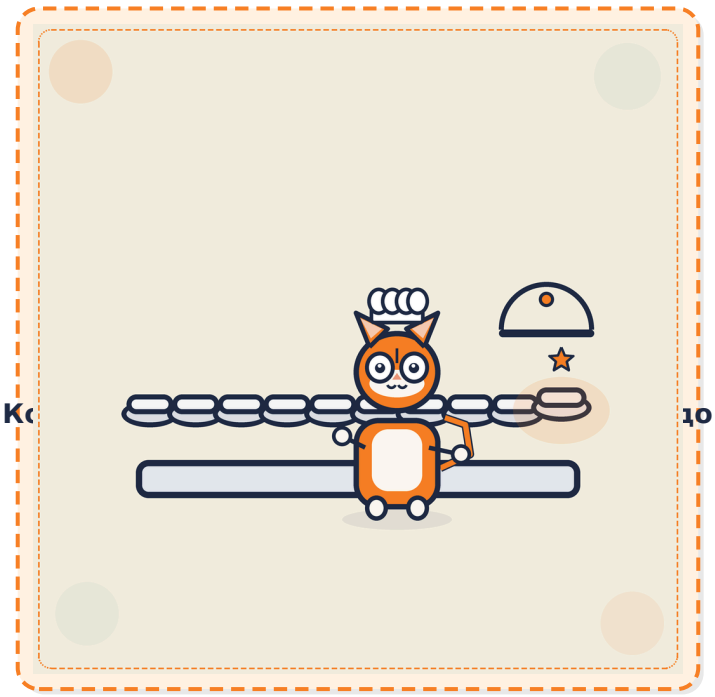
Глубже — в реальные данные, в доменную сложность, в комплаенс, в юридические ограничения, в накопленный production-опыт, в traces и eval-наборы, в локальную интеграцию с чьей-то реальной операционной жизнью.

Отсюда жёсткое следствие. Самое опасное место для стартапа сегодня — быть тонкой прослойкой посередине, которую следующая большая

модель легко съест как фичу. **Тонкая прослойка — это домик, построенный на будущей трассе железной дороги. Пока поезда нет, вы живёте. Когда приходит поезд — обсуждать уже нечего.** Строить безопаснее либо выше, либо глубже. Посередине остаётся всё меньше воздуха.

13

Вкус становится экономическим активом



Когда все научились готовить прилично, ресторан начинает продавать не еду, а выбор. Выбор того, что вообще стоит подавать. Выбор того, что лучше выкинуть из меню. Вкус.

AI поднимает пол мастерства. Мир начинает заполняться адекватными, нормальными, средними решениями — в количестве, которого раньше индустрия не видела. **Тут дорожает не способность сделать. Дорожает способность выбрать, что вообще стоит делать.** Вкус перестаёт быть приятным бонусом. Он становится активом, который конвертируется в деньги.

Вкус в этом контексте — это не про эстетику. Это про способность выбрать правильную проблему. Отрезать лишнее. Не увязнуть в усреднённом. Понять, что действительно важно пользователю, а что только кажется важным. Правильно расставить приоритеты. Не строить лишнюю сложность.

Если среднее стало дешёвым, то дорогим становится хороший выбор.

14

Старая карьерная лестница тихо умерла

Существует лестница, по которой в индустрии привыкли подниматься: джун, миддл, сеньор, стафф, принципал. Эта лестница была построена на одной валюте — часах, проведённых руками в коде. Больше написал — выше поднялся. Дольше стаж — крепче позиция.

Эта валюта подешевела в десять раз. Значит, и лестница подешевела в десять раз.

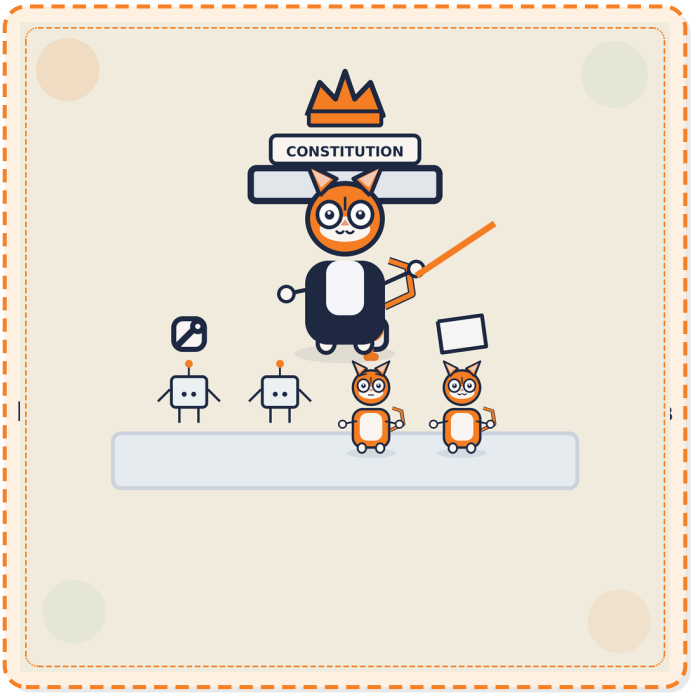
Десять лет опыта в написании CRUD-приложений больше не конвертируются в сеньорство автоматически. **Новая карьерная траектория меряет другое — не стаж, а глубину суждения.** Умеешь ли ты сказать «нет» правильному на вид решению? Умеешь ли за пять минут понять, где этот модуль сломается через полгода? Умеешь ли отличить красивую архитектуру от работающей? Умеешь ли спроектировать постановку задачи так, чтобы шесть разных исполнителей поняли её одинаково?

Этот навык не накапливается по годам. **Он накапливается по количеству правильно пойманных плохих решений.** Поэтому на глазах у нас меняется и наём, и продвижение, и оценка. Старые сигналы перестают работать. Новые ещё не устоялись. В этой промежуточной

пустоте будет болезненно многим — и выгодно тем, кто рано заметил, что валюта сменилась.

15

Человек становится губернатором системы



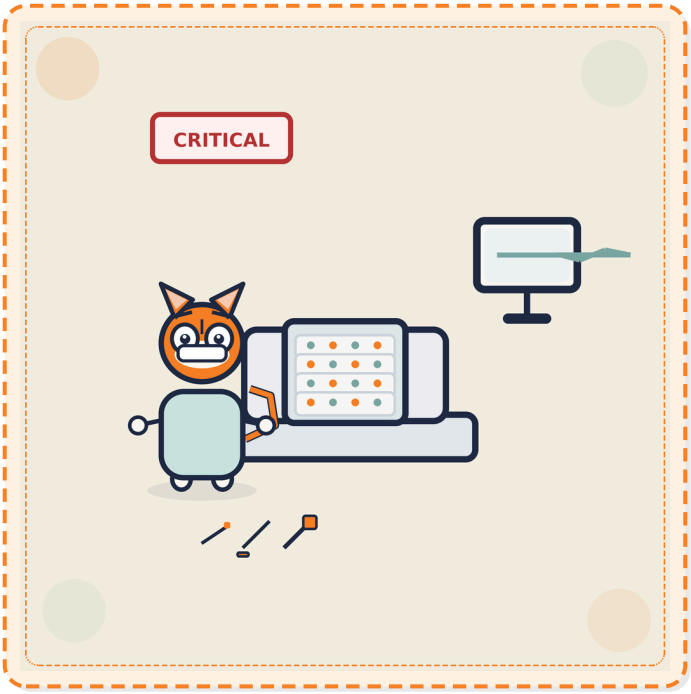
Сильный человек индустриального века был похож на кузнеца: он сам бил по железу, сам ковал каждую деталь, и его ценность жила в руках. **Сильный человек AI-эпохи ближе к дирижёру. Дирижёр не играет ни на одном инструменте. Но без него оркестр превращается в шум.**

Самый ценный человек теперь — не тот, кто вручную кладёт каждый кирпич. Это тот, кто задаёт смысл. Определяет границы доверия. Различает критичное и вторичное. Правильно распределяет задачу между разными классами решений — где нужна модель, где классификатор, где человек, где жёсткое правило. Умеет держать систему в русле. Удерживает связность, когда всё вокруг ускоряется.

Человек будущего не строит каждый кирпич вручную. Он задаёт конституцию системы. Он — губернатор, а не исполнитель. И это не понижение роли. Это повышение её масштаба.

16

Где граница этой философии



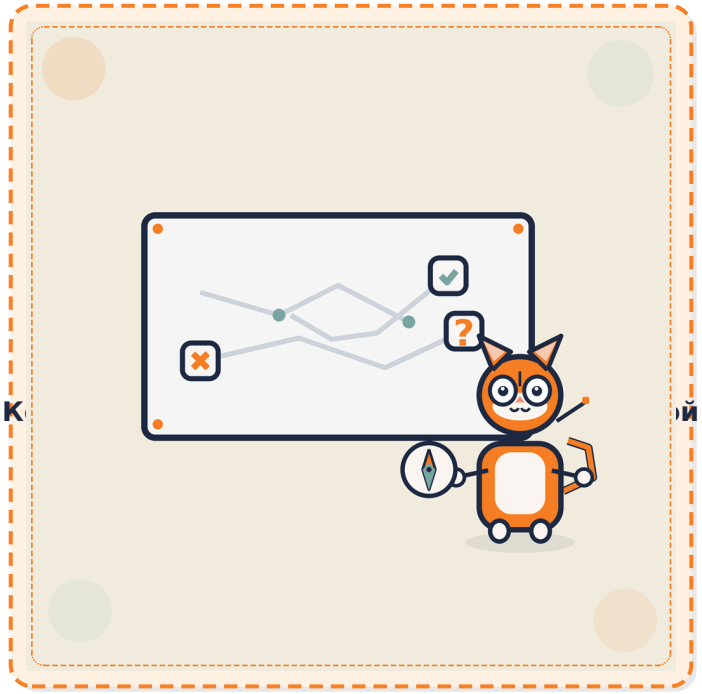
Всё вышесказанное легко принять за манифест хаоса. Это было бы неверным прочтением. Поэтому закончу честным ограничителем.

Есть классы систем, где цена ошибки высока: медицина, безопасность, критическая инфраструктура, платёжные ядра, промышленные контуры — места, где один сбой может реально дорого стоить человеческой жизни или деньгам, которые невозможно вернуть. **Туда эпоха серых ящиков приходит не как отмена строгости, а как усиление требований к контурам доверия.** Поисковая фаза становится быстрее. Несущая часть искусством не становится — она остаётся инженерией, просто инженерией другого уровня.

Не нужно романтизировать хаос. Нужно научиться локализовать хаос там, где он полезен, и не выпускать его туда, где он разрушителен. У свободы в инженерии всегда есть адрес прописки. Выбор этого адреса и есть профессиональное суждение.

17

Главная формула



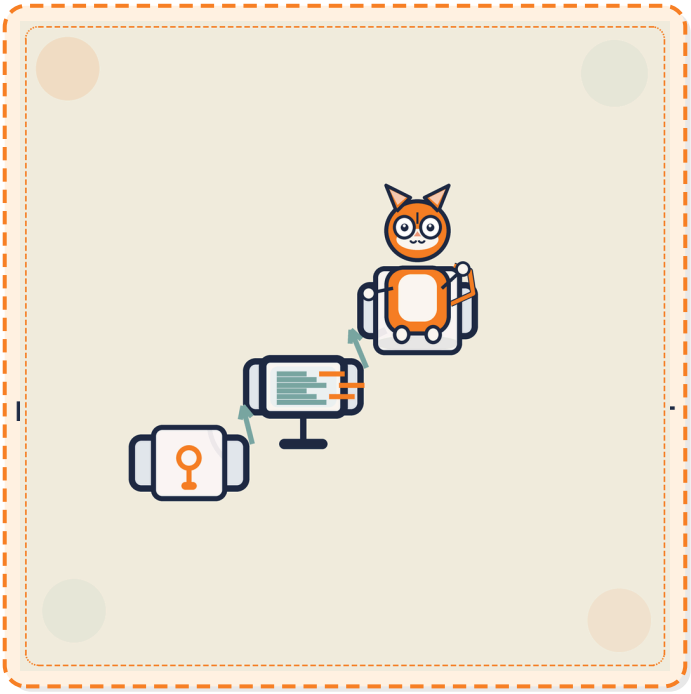
Если всё это сжать в одну мысль, она звучит так.

В AI-эпоху разработка и компания перестают быть фабрикой по изготовлению кода — и становятся системой управляемого поиска.

Код дешевеет. Варианты дешевеют. Нижние слои коммодитизируются. Дорожает другое: верификация, сжатие смысла, скорость получения сигнала, скорость обратимых виражей, снижение энтропии, вкус, дистрибуция, доверие, способность удерживать стабильный внешний продукт при очень текучей внутренней машине.

Финал

Настоящий вопрос эпохи



Новая эпоха не требует от нас понимать всё до последнего винтика. Она требует другого.

Уметь работать с серыми ящиками. Уметь задавать контекст вместо псевдоточности. Уметь быстро проверять гипотезы — и ещё быстрее закрывать ложные. Уметь поворачивать там, где это обратимо, и держать курс там, где поворот невозможен. Уметь не захлебнуться в собственных вариантах. Уметь строить компании вокруг реальной ценности, которую нельзя съесть следующим релизом большой платформы.

Главный вопрос ближайших лет звучит уже не так, как он звучал десять лет назад. Не «кто лучше пишет код». А — «кто быстрее превращает намерение в проверенное, надёжное поведение системы».

Вот это и есть настоящий вопрос эпохи.

★

Ударные фразы для расстановки по тексту

Их можно вынести в слайды, на обложку, в заставки, в твиты. Каждая держится сама по себе.

01	**Дефицит не исчезает. Он переезжает.**
02	**Генерация стала дешёвой. Дорогой стала верификация.**
03	**Инженер больше не автор. Он судья.**
04	**Код — не актив. Код — это грязная посуда после ужина.**
05	**Мы больше не собираем мир из белых ящиков. Мы управляем серыми.**
06	**Инженерия не умерла. Она переехала.**
07	**Плохой ТЗ-шник прописывает шаги. Хороший — прописывает инварианты.**
08	**Разные исполнители могут разойтись в реализации. Но не в замысле.**

09 **Дешёвый инструмент требует более дорогой дисциплины мышления.**

10 **Двести метрик и ни одного решения — это не приборная панель. Это витрина.**

11 **В системе есть горячие регионы и замороженные. Перепутать их — клиническая смерть.**

12 **Виражи без страховки — не смелость. Гонка к обрыву.**

13 **Внутри жидкая. Снаружи твёрдая.**

14 **Энтропия — не событие. Это погода.**

15 **Тонкая прослойка посередине — это дом на будущей трассе железной дороги.**

16 **Если среднее стало дешёвым — дорогим становится хороший выбор.**

17 **Десять лет опыта в CRUD больше не делают из вас сеньора.**

18 **Человек будущего не кладёт кирпичи. Он задаёт конституцию системы.**