Вайб-кодер надеется на результат

Агентный инженер его гарантирует

Никита Гурняк · flutter teamlead, 2ГИС

Май 2026

Меня заменил ИИ.

Теперь у меня есть время на доклады.

Лестница зрелости

лестница зрелости

§

УРОВЕНЬ	ОПИСАНИЕ	ХАРАКТЕРИСТИКА
0	Вайб-кодинг	Оно как-то работает
1	Не используем ИИ	Зато своё, с огорода
2	Умный автокомплит	Экономим время или отвлекаемся?
3	ИИ-assisted программист	Делегируем бойлерплейт
4	Агентный инженер	Команда агентов, которые приносят результат

> Вайб-кодер **надеется** на результат, агентный инженер его **гарантирует**.

Ландшафт инструментов

как люди работают с моделями

§

СПОСОБ	ПЛЮСЫ	МИНУСЫ
Vibe SaaS	Делает задачи, прячет сложности	Не расширяемо, умирает после mvp
Веб-чат	Beginner friendly, замена гуглу	Нет связи с проектом
IDE extension	Видит контекст	Привязан к IDE
CLI	Параллелизация, автоматизация, компоновка	Порог входа

cli — the juice

§

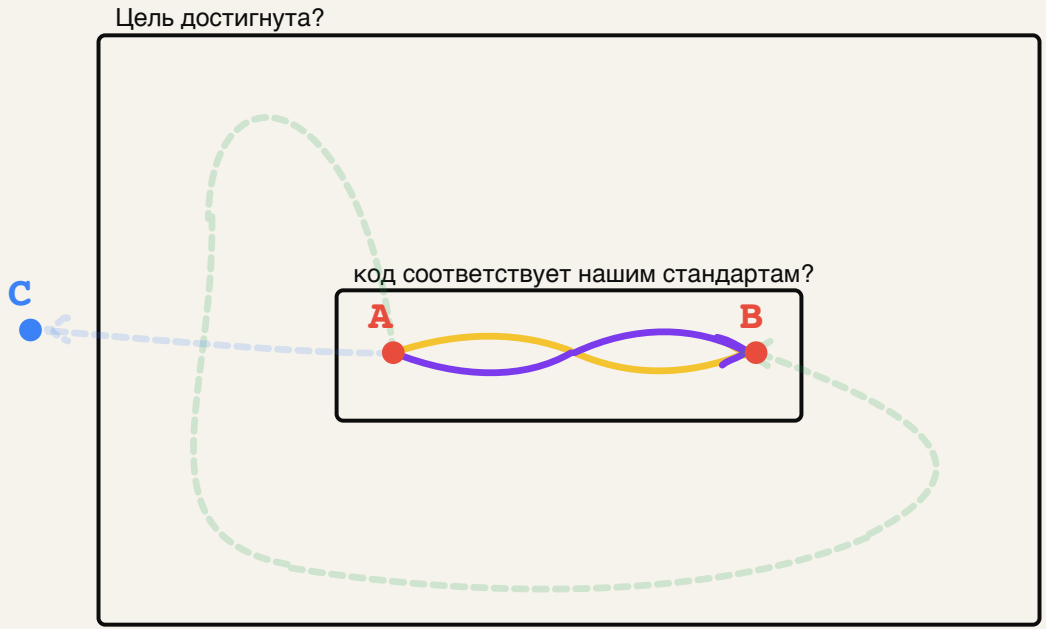
Почему CLI:

- Скриптуется
- Параллелизация
- Универсальность: код, ресёрч, документация…
- Компоновка пайплайнов произвольной сложности
- CI / внешние триггеры на твоих условиях

Ремарка:

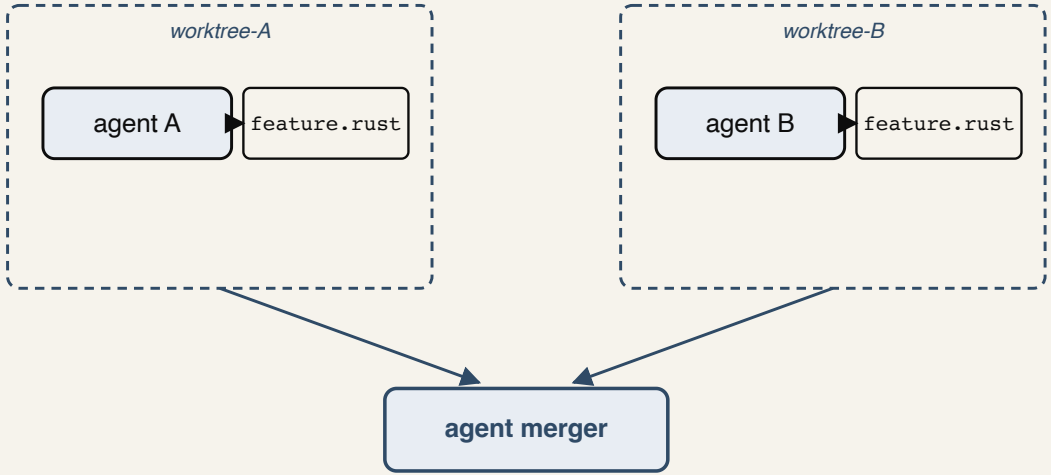
- Cursor = гибрид CLI + IDE
- Плагины сдвигаются в сторону агентики
- Но CLI — фундамент для продвинутых пайплайнов
- Anthropic строит Vibe-экосистему вокруг своих продуктов

от хаоса к гарантии



путь стохастичен · граница детерминирована

изоляция масштабирует



параллелим попытки, не код

Из чего состоит агент?

Агент стоит на четырёх столпах

четыре составляющих агента

§

01 **LLM** — мозг агента: Claude, GPT, локальные модели

02 **Промпт** — что ты *говоришь* агенту

03 **Контекст** — что агент *знает* в этот момент

04 **Инструменты** — как агент действует в мире

> В сумме — **агент**.
> Лучше любой из четырёх = **лучше выполненная работа.**

промпт ≠ контекст

§

Промпт — что ты говоришь агенту
Контекст — что агент знает в этот момент

Промпт

- Дискретный, авторский артефакт
- Команды, спеки, шаблоны
- Единица намерения

Контекст

- Динамический, собираемый
- Код, доки, логи, история
- Инфраструктура вокруг промпта

> Один промпт в разных контекстах → разные результаты.
> **Prompt engineering — подмножество context engineering.**

Отдельно про LLM

модель - не ограничение

§

Сырая **интеллектуальность** модели — больше **не ограничение**.*

Что **остаётся** ограничением:

- Контекстное окно
- Сложность (структура) кодбазы
- **Твои** способности как инженера

* — речь про sonnet 3.7+ и сопоставимые. Локальные тоже идут — но не в этом докладе.

Посмотрим в деле?

осмотр кодбазы до

Setup: `mobius-conf-demo` @ `tag-0-baseline`. Простая задача, спагетти-код внутри.

Что это

Машины POST хук-события Claude Code в backend.
SQLite пишет, WebSocket стримит, Vue показывает фид.

```
mobius-conf-demo/
├── server/   FastAPI + SQLAlchemy + SQLite
├── client/   Vue 3 + TypeScript + Vite
└── scripts/  emit_fake_events.py
```

~600 LOC. Один POST, один WebSocket, одна таблица.

server/utils/parse.py

```python
def process_incoming_event(raw, db=None):
    if isinstance(raw, str):
        try:
            data = json.loads(raw)
        except:
            data = {}
    elif isinstance(raw, dict):
        data = raw
    evt = data.get("event_type") or data.get("type")
    if evt is None:
        e = "Unknown"
    else:
        e = str(evt)
    # ...ещё 70 строк
```

tag-0-baseline

действуем интуитивно

§

Setup: та же кодбаза, обычный prompt без подготовки, без настроенных разрешений. Считаем Attempts и Presence.

tag-0-baseline · naive run

Как выжать максимум из агентов

1. встань на место агента

§

Агент каждый раз начинает с **чистого листа**.

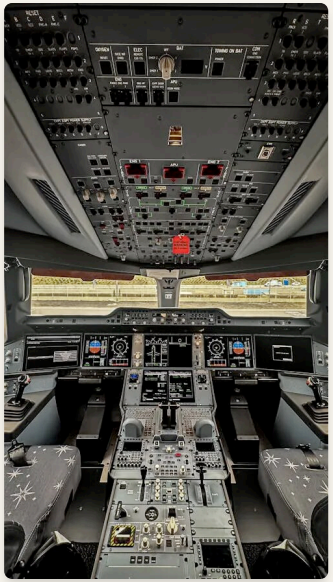
Представим **твой онбординг** на проект:

- Какие инструменты используются?
- Командные практики?
- Внешние ресурсы?
- Зачем вообще существует проект?

> Сложно тебе? Сложно и агенту.

1. встань на место агента — demo

Setup: пишем `CLAUDE.md` — стек, слои, правила, naming. Онбординг для агента.



«V1… подъём… положительный набор, шасси убрать.»

```
## Where things live

Strict layering. New code goes to one of:

- `server/routes/`    — HTTP endpoints. Thin.
- `server/services/` — business logic. Side effects explicit.
- `server/utils/`     — pure helpers. No I/O, no DB.
- `server/models.py` — SQLAlchemy models only.

If you can't place new code into one of these,
the rule needs a new line — flag it, don't improvise
```

tag-0 → tag-2 · CLAUDE.md

2. дай агенту ошибиться и исправиться

§

Цикл:

01 Запрос

02 Валидация

03 Исправление

04 → Повторить

Агент **будет** ошибаться.
Тесты ловят ошибки рано.

> Перестань брать ответственность за тестирование. Учи агентов делать это за тебя.

Рычаги:

- Логгер
- Линтер
- Git hooks
- Тесты
- Компиляция

Да, компиляция бывает небыстрая. Но машинное время масштабируется, моё — нет.

+ Логи и трейсы — агент видит, что реально произошло. Логи говорят «не так». Трейсы — «где».

2. дай агенту ошибиться и исправиться — demo

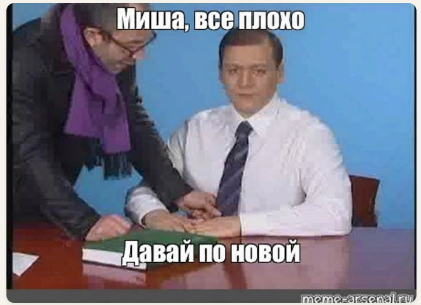
§

Setup: добавляем structlog + request_id middleware. Прошу починить баг — агент читает логи сам.

server/utils/notifications.py

```python
# до · tag-2
try:
    payload = _build_payload(target)
    loop.create_task(_send_to_all(json.dumps(payload)))
except:
    pass

# после · tag-3
try:
    payload = _build_payload(target)
    loop.create_task(_send_to_all(json.dumps(payload)))
except Exception:
    log.exception("broadcast.failed", event_id=target.id, event_ty
```



тишина → сигнал · агент читает логи и чинит сам

tag-2 → tag-3 · observability

3. научи агентов решать по-своему

§

Научи агентов решать по-своему

Два акцента:

01 **НАУЧИ** — заворачивай в скилл
Паттерны и инструменты становятся переиспользуемой единицей.

02 **ПО-СВОЕМУ** — твои паттерны, твои инструменты
Скилл несёт *как ты думаешь* и *чем ты решаешь*.
Агент решает задачи **как ты**.

> Это один из **обязательных** шагов к решению проблемы «моя кодбаза слишком большая для ИИ».

3. научи агентов решать по-своему — demo

Setup: `.claude/skills/tdd/SKILL.md` — паттерн «думай как валидировать → потом пиши»

```
---
description: Use this skill when implementing any
function with business logic. Captures red-green:
define validation before writing implementation.
---

# Purpose

You write tests before code. Tests describe intent;
implementation has to match intent.

## Workflow

1. List 3–5 edge cases that would break a naive impl.
2. Write a pytest function for one case.
3. Run pytest. The new test MUST fail.
4. Write the smallest implementation that passes.
```

tag-3 → tag-4 · tdd skill

3. научи агентов решать по-своему

Setup: `.claude/skills/pinchtab/` — браузер обёрнут в скилл. Любой твой инструмент можно так же.

```
Every PinchTab automation follows this pattern:

1. Ensure the right server / profile / instance.
2. Navigate with `pinchtab nav <url>`.
3. Observe with `snap -i -c` — collect refs (e5, e12).
4. Interact: `click`, `fill`, `type`, `press`.
5. Re-snapshot after any DOM-changing action.
```

tag-4 → tag-5 · pinchtab skill

Доверие идёт с малого

доверие

§

Когда агент начинает решать задачи так, как это делаем мы, — постепенно можно снижать вовлечённость в перекидывание сообщениями.

Агент это результат решения **класса** задач. Но этот класс решает *Agentic engineer*.

Мы дошли до точки, когда можно немного отпустить вожжи.

Agentic loop

in-loop и out-loop

§

In-loop

Ты в цикле: промптишь, смотришь результат, итерируешься.

Агент — инструмент в твоих руках.

Out-loop

Агент работает автономно по триггеру. Приносит код пока ты размышляешь над направлением.

Агент — автономный исполнитель.

> Для хорошо настроенной агентики **бутылочное горлышко — человек**

in-loop и out-loop — demo

Setup: тот же агент, два режима запуска. Где граница доверия — там граница между in и out.

In-loop · `claude`

```
$ claude
[?] Bash(uv run pytest)          y
[?] Edit routes/events.py        y
[?] Edit services/feed.py        y
[?] Bash(uv run pytest)          y
[?] Edit tests/test_feed.py      y
...
```

approve · edit · repeat

Out-loop · `claude -p "..."`

```
{
  "permissions": {
    "allow": [
      "Bash(uv run *)",
      "Bash(bun run *)",
      "Bash(git diff *)",
      "Edit(server/**)",
      "Edit(client/**)"
    ]
  }
}
```

fire · forget · review diff

tag-5 → tag-6 · out-loop config

4. сфокусируй агентов

§

Один агент, один промпт, одна цель.

- Огромные контекстные окна → рассеянные агенты
- Сфокусированный инженер = продуктивный, то же с агентами
- LLM — prediction machine. Узкий контекст → лучше предсказания

> Спеки изолируют задачи — идеальный инструмент фокусировки.

Историческая параллель

делегирование — не новая идея

§

- **Римская черепаха** — простые солдаты, сложная формация, командиры разных уровней
- **Система вассалов** — цепочка делегирования, дробление ответственности
- **Звания в армии** — каждый уровень = свой масштаб задач
- **IT** — менеджеры → лиды → разработчики → …

Общий паттерн:

- Одно «железо» — человек
- Разные «программы» — каждый уровень отвечает за свой масштаб
- Ориентир на результат

4. сфокусируй агентов — demo

Setup: `/orchestrate` раздаёт задачу трём агентам — у каждого свой набор tools.

`.claude/agents/{plan,implement,verify}.md`

Роль	Tools	Что делает
plan	Read, Grep, Glob	читает код → пишет спеку
implement	Read, Write, Edit, Bash	TDD по спеке
verify	Read, Bash, Grep	тесты + acceptance

tag-6 → tag-7 · orchestrator

5. оптимизируй кодбазу для агентов

§

Бонус: **хорошо для агента = хорошо для новичка:**

- Чёткие точки входа, константы, типы
- Говорящие имена (Information Dense Keywords)
- Файлы < 1K строк, 1 ответственность на файл
- Консистентность побеждает сложность

5. оптимизируй кодбазу для агентов — demo

§

Setup: изолируем side-effects. `server/routes/events.py · ingest` — линейный pipeline вместо скрытых listeners.

До · tag-7

```python
async def ingest(payload, db):
    # ... auth + logging
    event = process_incoming_event(payload, db=db)
    return {"ok": True, "id": event.id}

# где-то ещё, в utils/notifications.py:
@event.listens_for(Event, "after_insert")
def broadcast_to_websocket(...): ...

@event.listens_for(Event, "after_insert")
def update_session_summary(...): ...
```

После · tag-8

```python
async def ingest(payload, db):
    # ... auth + logging

    event = parse_event(payload)
    store_event(event, db)
    broadcast_event(event)
    update_session_summary(event, db)

    return {"ok": True, "id": event.id}
```

tag-7 → tag-8 · side-effects isolated

6. уделяй время агентике

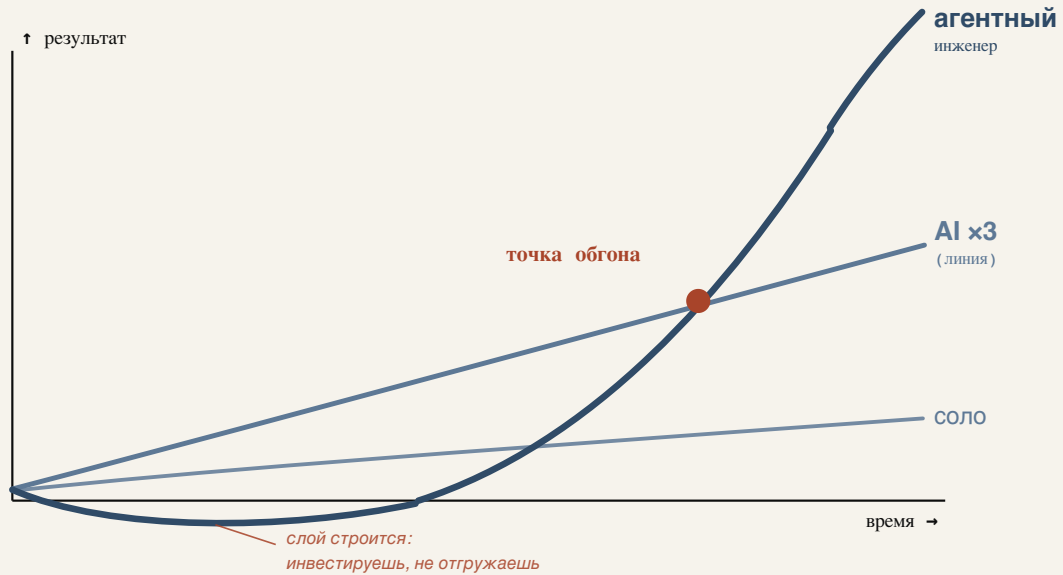
§

Два слоя:

- **Application layer** — кодбаза, продукт
- **Agentic layer** — промпты, шаблоны, скрипты, спеки, доки

Agentic layer это инвестиция с **экспоненциальной отдачей**:
вложил → выиграл время → реинвестировал.

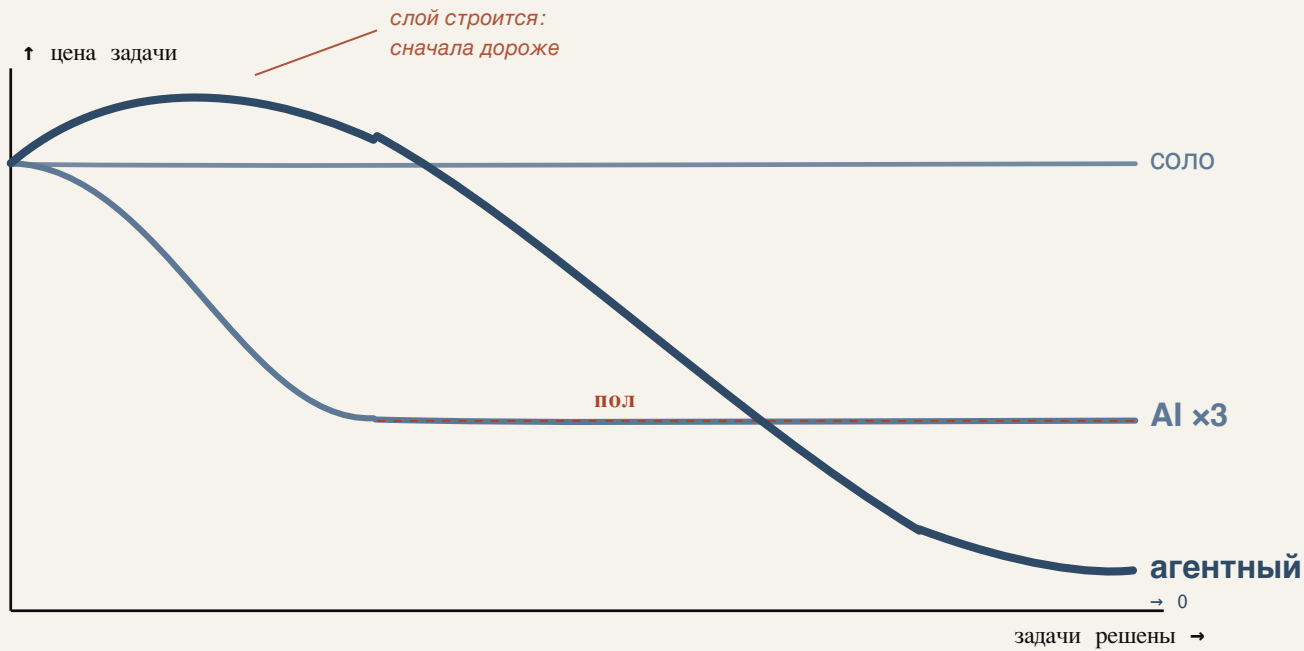
6. уделяй время агентике — почему?



множитель ≠ экспонента

Я сейчас работаю над агентикой или над приложением?

6. множитель упирается в пол



множитель упирается в пол. компаундер — нет.

7. настрой внешние триггеры

§

ADW — *Agentic Developer Workflow*. Агента дёргает **событие**, а не ты руками.

Откуда дёргать:

- GitHub / GitLab / Forgejo — issues, PR, комментарии
- Jira / Linear — переход в статус, новый тикет
- Cron / launchd — по расписанию
- File / queue watcher — изменения в S3, в очереди

> Цель: **Presence → 0**, **Attempts → 1**.

Term ADW — спасибо andyxchan.medium.com

7. настрой внешние триггеры — demo

Setup: GitHub label `agent-go` → workflow → `/orchestrate` → PR. Меня в цикле нет.

.github/workflows/agent-trigger.yml

```yaml
on:
  issues:
    types: [labeled]

jobs:
  dispatch:
    if: github.event.label.name == 'agent-go'
    steps:
      - uses: actions/checkout@v4
      - name: Dispatch orchestrator
        run: uv run scripts/issue_to_agent.py
```

> Меня нет ни в одном шаге.

tag-8 → tag-9 · external triggers

Круто!

Круто, но… есть нюанс

самый сложный триггер — внутренний

§

Webhook слушает. Cron настроен. Worktree готов.

И всё равно я открывал терминал.

И все равно я поглядывал за агентом.

> Out-of-loop — **не** инженерная задача.
> Это **разрешение перестать смотреть.**

Чем лучше инфра — тем заметнее, что узкое место уже не код.

внутренний триггер

§

"Доверие начинается с малого"

All-in-one сетапы с интернета были слишком большими чтобы вверится в них

Это определило **мой майндсет**

Майндсет

майндсет

§

Если агент делает не то — значит, я его не так настроил.

Раз косяк **мой** — значит **могу улучшить.**

Система **улучшаема** — стоит в неё инвестировать.

Речь о моделях сопоставимых или превосходящих claude sonnet 3.7.

а зачем делаем то что делаем?

§

Магия не в лампе, а в **формулировке**.

Джинн **исполняет** желания, а не придумывает их.

А вот **LLM** вполне себе **придумывает** и может делать не то, что ты имеешь в виду.

Пойми, что ты хочешь

пойми, что ты хочешь

§

Можно построить завод по производству спорткаров, но тебе нужен КамАЗ для перевозки щебня.

> Время на спеку = время в правильную имплементацию.
> Большинство инженеров тратят его недостаточно.

Не можешь написать спеку — напряги агента, он поможет.

Баланс между *душно* ↔ *и так сойдёт* — ловится на опыте.

пойми, что ты хочешь

§

> Спека это **не** про формализм
> это про то, чтобы **ты** сам **понял** что имеешь в виду

кейсы из индустрии

§

CCC

C-компилятор на Rust
100K LOC
Человек писал только тесты

Closing the loop

anthropics/claudes-c-compiler

Devin + Nubank

Миграция ETL-монолита
18 мес → 2 мес
12x эффективность

devin.ai/customers/nubank

Stripe Minions

1000+ PR/неделю
Полностью автономно
400+ MCP tools

stripe.dev/blog

Как понять что улучшаемся? KPI

kpi агентной работы

§

Максимизируй ↑

- **Size** — tool calls, единица результата
- **Streak** — серия 1-attempt промптов подряд

Минимизируй ↓

- **Attempts** — "отправка промпта", сколько раз вмешивался для результата
- **Presence** — твоих действий в процессе

Больше tool calls = больше impact.

> Время инженера дорого, время машины дешево.

Кейс 1 — Реальный масштаб

кейс 1 — реальный масштаб

Кодбаза:

- 200K+ LOC, complexity 14K+
- Нулевой agentic layer на старте
- проблема чистого листа

Что вложил:

- ~6 часов написание фичи руками
- ~7 часов на создание слоя агентики
- Тот же подход:
  - оптимизируем 4 столпа
  - идём в out-loop

Результат:

- Тонкая спека → end-to-end PR
- Вторая фича того же "калибра" — ~1 час для агента и 10 минут моего времени
- $88 за тот день в Claude Code

кейс 1 — вывод

§

> Если агент делает не то — значит, я его не так настроил.

кстати, подписка за $200 даёт токенов на ~$2700

Кейс 2 — Ежедневная выгода

кейс 2 — ежедневная выгода

§

Сбор и анализ контекста по проекту, кросс-валидация источников (jira, mattermost, confluence, gitlab), подготовка репорта.

Занимало 1–2 часа в день. Сейчас <20 минут, на протяжении нескольких месяцев.

Только моего времени экономит на полтора рабочих дня в месяц. Но настоящая ценность в том, что снижает когнитивную нагрузку и помогает предотвращать человеческий фактор и мисинтерпретацию.

кросс-домен

§

Полезные домены, улучшая себя в которых, можно получить ~~не~~ожиданный прирост в агентике:

- Личный трекинг тасок/целей
- Ведение базы знаний
- Навык делегировать
- Умение объяснять что ты хочешь

Субъективно, не задерживаемся.

Итоги

что забрать с доклада

§

01 Агентика — делегирование и порядок, не магия

02 Четыре столпа: модель, промпт, контекст, инструменты

03 In-loop → out-loop

04 Framework:

- Встань на место агента
- Научи агентов решать по-своему
- Дай агенту ошибиться и исправиться
- Один агент — одна задача
- Инвестируй в agentic layer
- Строй систему, которая строит системы
- Настрой внешний триггер

Vibe coder < Engineer < Agent engineer

Масштабируйте себя.

demezy.com

Демо репозиторий

TG @demezy