СТУДИЯ NOTEVIBE · ИНЖЕНЕРНЫЙ ВАЙБКОДИНГ

# После вайба

Как довести проект, собранный с ИИ, до состояния, за которое не стыдно

**Сергей Кожуховский**

*в соавторстве с ИИ-агентом*

NOTEVIBE.RU

# Часть 0. Первый вечер

Эта часть занимает один вечер. К её концу у вас будет работающая вещь: страница, которая принимает обращения, сохраняет их и показывает списком. Небольшая, без украшений, но настоящая – с хранением, проверкой и инструкцией запуска.

По дороге случится ещё одно, и оно важнее самой поделки. Примерно на середине вечера вы получите от агента результат, который выглядит готовым, покажете его себе, порадуетесь – а потом обнаружите, что он врёт. Мы это подстроили. Разбираться, как такое возможно и что с этим делать, вы будете всю оставшуюся книгу, но пережить это стоит один раз самому. Чужие ожоги не учат.

Ничего устанавливать заранее не нужно, кроме того, что мы поставим вместе. Опыт программирования не требуется. Требуется два-три часа и готовность выполнять шаги по порядку, без забеганий вперёд.

## 0.1. Двадцать минут до первой страницы

Соберём рабочее место. Его три части: язык, на котором будет написан проект, папка проекта под присмотром системы контроля версий и агент, которому мы будем ставить задачи.

Языком будет Python. Причина выбора приземлённая: он ставится одной командой, ничего не требует сверх себя, и всё, что мы соберём, запустится на любой машине – Windows, Mac, Linux. Это позволяет нам пообещать: у вас получится то же, что у нас. Обещание дороже рассуждений о том, какой язык лучше.

Откройте терминал и проверьте, что Python есть:

```
python3 --version
```

Если в ответ пришёл номер версии – всё на месте. Если команда не найдена, поставьте Python с официального сайта; на Windows команда называется `python`, без тройки. Это единственная развилка на весь вечер.

Теперь папка проекта:

```
mkdir zayavki
cd zayavki
python3 -m venv .venv
source .venv/bin/activate
pip install fastapi uvicorn python-multipart
```

Три последние строки заводят так называемое окружение – отдельный ящик с инструментами именно для этого проекта, чтобы он не перемешивался с остальной машиной, – и кладут в него три инструмента: FastAPI, на котором пишется серверная часть, uvicorn, который её запускает, и python-multipart, без которого не будут работать формы. На Windows строка активации другая: `.venv\Scripts\activate`.

Про третий пакет стоит сказать отдельно, потому что это первый урок вечера. Формы – настолько обычное дело, что ожидаешь их работы «из коробки». Но FastAPI не станет разбирать отправленную форму, пока не найдёт python-multipart, и сообщит об этом только в момент запуска. Мы узнали это ровно так же, как узнаете вы: запустив. Никакое чтение документации по диагонали такие вещи не выявляет – их выявляет прогон.

Дальше – система контроля версий. Сначала скажем ей, чего запоминать не надо, иначе в историю проекта уедет папка с инструментами и файл базы с данными. Создайте файл `.gitignore`:

```
.venv/
__pycache__/
*.db
```

И заводите хранилище истории:

```
git init
```

Одна команда. Git запоминает состояния проекта: каждый раз, когда вы фиксируете рабочую точку, к ней можно вернуться. Пока вы работаете один и всё идёт хорошо, git кажется лишним. Он перестаёт казаться лишним в тот момент, когда агент переписал половину проекта и стало хуже, а вчерашней версии больше нет. У нас в практике был случай ещё неприятнее: две рабочие сессии правили одну папку без git, и одна молча затёрла работу другой – на опубликованный сайт ушёл чужой вариант страницы. Заметили по факту, восстанавливали по обрывкам. С git это стоило бы одной команды.

Теперь первая страница. Создайте файл `main.py`:

```
from fastapi import FastAPI
from fastapi.responses import HTMLResponse

app = FastAPI()

@app.get("/", response_class=HTMLResponse)
def home():
    return "<h1>Приём обращений</h1><p>Сервис работает.</p>"
```

И запустите:

```
uvicorn main:app --reload
```

Откройте в браузере адрес `http://127.0.0.1:8000`. Если на экране заголовок «Приём обращений» – у вас работает собственный сервер. Он живёт только на вашей машине и виден только вам, но это уже не картинка: браузер отправил запрос, программа его обработала и ответила.

Зафиксируем рабочую точку:

```
git add .
git commit -m "Пустое приложение запускается"
```

Слово «коммит» дальше будет встречаться часто – это и есть зафиксированное состояние проекта, точка, к которой можно вернуться. Правило вечера: коммит после каждого работающего шага. Привычка «зафиксирую потом разом» не срабатывает никогда.

> **Если вы заказчик.** Вы не обязаны выполнять эти команды – но вопрос «покажите репозиторий» стоит выучить. Репозиторий – это история проекта: что менялось, когда и в каком порядке. Если подрядчик не может его показать, у проекта нет истории, а значит, нет ни защиты от потери работы, ни возможности разобраться, кто и что сломал. Вы платите в том числе за это.

На часах должно быть минут двадцать. Идём к агенту.

## 0.2. Форма, которая врёт

Теперь поставим первую задачу агенту. Какому именно – решите сами: подойдёт любой из инструментов, где модель пишет код по описанию. Слова договоримся использовать так: модель – сама нейросеть, агент – инструмент, в котором она умеет работать с файлами вашего проекта. Мы сознательно не называем в книге конкретные продукты и версии: они меняются быстрее, чем печатается тираж. Актуальный список с нашими пометками живёт на странице книги: notevibe.ru/kniga.html – там он обновляется, с датой сверки у каждой строки.

Задача звучит так. Скопируйте её целиком:

```
Собери форму приёма обращений для страницы на FastAPI.
Поля формы: imya, kontakt, tekst. Отправка методом POST на
/otpravit.
После отправки человек должен видеть подтверждение, что
обращение принято.
Сделай минимально: одна страница, без оформления, без
регистрации.
```

Имена полей и адрес отправки мы задали сами, и это не придирчивость. Дальше мы будем писать проверку, которая обращается к сервису по этим именам; если каждый раз позволять агенту выбирать их на свой вкус, проверки придётся переписывать после любой правки. Названия – часть договора с проектом, и договор пишет владелец.

Агент вернёт код – скорее всего, охотно и быстро, с формой, кнопкой и сообщением «Спасибо, ваше обращение принято». Вставьте его результат в проект, как он предложит, перезапустите, откройте страницу.

Заполните форму. Нажмите кнопку. Прочитайте «Спасибо».

Приятный момент. Форма выглядит как настоящая, отвечает как настоящая, и внутри у вас, скорее всего, возникло ощущение «почти готово». Запомните это ощущение – мы будем к нему возвращаться.

Теперь обновите страницу и попробуйте найти отправленное обращение.

Его нет. Не в списке – списка тоже нет. Не в файле – файла не появилось. Обращение не легло никуда: сообщение «Спасибо» было нарисовано в момент нажатия кнопки, и на этом всё закончилось. Если бы такой формой пользовался настоящий человек с настоящей проблемой – его обращение исчезло бы молча, а он остался бы с уверенностью, что его услышали.

Здесь важно не проскочить мимо главного. Агент не ошибся и не обманул. Мы попросили форму с подтверждением – мы её получили. Про то, что обращение должно где-то сохраниться, в задаче не было ни слова, и агент заполнил пропуск самым дешёвым способом: никак. Модель охотно достраивает недосказанное, и достраивает правдоподобно. Правдоподобно – это про внешний вид, у которого нет обязательств перед внутренним устройством.

Такое случается не только с новичками. В части III мы разберём, как эта же поломка несколько месяцев прожила у нас, и почему её не заметила ни одна проверка.

Отсюда – главный инструмент этой книги. Шесть вопросов, которые задаются любому результату генерации, пока не станут рефлексом:

1. Где хранятся данные?
2. Что останется после обновления страницы?
3. Что произойдёт при ошибке – и увижу ли я её?
4. Какие действия на самом деле уходят на сервер?
5. Что увидит человек, если заполнит форму неправильно или не заполнит вовсе?
6. Смогу ли я запустить это по инструкции, на другой машине, а не только в текущем окне?

Прогоните нашу форму по списку: первый же вопрос не имеет ответа, второй отвечает «ничего». Этого достаточно, чтобы понять уровень результата – картинка. Картинка полезна: её можно показать, по ней можно обсуждать, как это будет выглядеть. Просто она пока не продукт и не обязана им быть.

Зафиксируйте и это состояние тоже: `git commit -m "Форма без хранения – картинка"`. История проекта имеет право содержать неудачные точки, она для того и история.

## 0.3. Хранение: обращение перестаёт исчезать

Почин очевиден: обращению нужно место, где оно будет лежать. Поставим агенту вторую задачу – и обратите внимание, насколько она отличается от первой:

```
Обращения из формы должны сохраняться в базе SQLite.
Путь к файлу базы брать из переменной окружения ZAYAVKI_DB,
по умолчанию zayavki.db.
Таблица obrasheniya: id, imya, kontakt, tekst, sozdano, status
(status по умолчанию "новое").
Добавь страницу /spisok - список всех обращений из базы, новые
сверху.
После отправки формы - подтверждение с номером обращения.
Не переписывай проект целиком: измени только то, что нужно
для хранения и списка.
```

Четыре отличия делают эту постановку рабочей. Названо место хранения. Описаны данные – какие поля, какой статус по умолчанию. Путь к базе взят из переменной окружения, а не вписан в код: через полчаса это позволит проверкам работать со своей базой и не топтаться в настоящих обращениях. И проведена граница правки: последней строкой мы запретили агенту переделывать всё подряд.

Про границу стоит сказать прямо. Без этой строки агенты любят возвращать «улучшенный» проект целиком, где заодно переименовано, переставлено и сломано то, что работало. Границу правки мы будем ставить в каждой задаче до конца книги.

SQLite мы выбрали по той же логике, что и Python: это база данных в одном файле, без установки, без службы, без пароля – и при этом настоящая. Для учебного проекта и для доброй половины небольших рабочих сервисов её достаточно.

Примите результат, перезапустите, откройте страницу. Заполните форму. Теперь – ритуал, ради которого всё затевалось:

Обновите страницу. Откройте `/spisok`. Обращение на месте.

Остановите сервер целиком (Ctrl+C в терминале) и запустите заново. Откройте список ещё раз. Обращение на месте.

Второй шаг – не паранойя. Между «данные пережили обновление страницы» и «данные пережили перезапуск» лежит целый класс поломок: агент мог сохранить обращения в память работающей программы, и тогда список выглядел бы настоящим ровно до первого перезапуска. «У меня всё работало, а после перезапуска пусто» – одна из самых частых первых аварий вайбкодинга, и ловится она одной командой остановки. Проверка перезапуском теперь входит в ваш набор навсегда.

Пройдитесь по шести вопросам из 0.2 ещё раз. Теперь у первых двух есть ответы: данные в файле `zayavki.db`, обновление страницы их не трогает. Вопросы три и пять пока висят – к ним вернёмся. Это нормальное состояние: важно знать, что именно ещё не сделано, вместо ощущения, что «в целом готово».

```
git add .
git commit -m "Обращения сохраняются в SQLite, список
работает"
```

## 0.4. Первый тест: проверяет машина

Сейчас вы проверяете проект руками: заполнить, отправить, обновить, посмотреть. Для одного сценария это терпимо. Но проект будет расти, и с каждой правкой руки должны будут перепроверять всё, что работало раньше, – а руки ленивы и склонны верить, что «там-то ничего не сломалось». Именно на этой вере агент и ломает проекты: правя одно место, он задевает другое, а вы узнаёте об этом через неделю.

Выход – проверка, которую выполняет машина. Автотест: маленькая программа, которая делает то же, что делали ваши руки, и сравнивает результат с ожидаемым.

Первый тест напишем сами, без агента, и это принципиально. Тест – это формулировка того, что проект обязан делать. Если попросить агента написать тесты к готовому коду, он добросовестно зафиксирует то, что код делает сейчас, включая ошибки: кривое поведение станет узаконенным. Контракт пишет заказчик работы. В нашем проекте заказчик – вы.

Поставьте инструменты для тестов и создайте файл `test_zayavki.py`:

```
pip install pytest httpx2
```

```
import os
import pathlib

# База для проверок - отдельная, иначе тесты засорят рабочие
обращения.
# Переменную выставляем ДО импорта main: он читает её при
загрузке.
TESTOVAYA_BAZA = "test_zayavki.db"
os.environ["ZAYAVKI_DB"] = TESTOVAYA_BAZA
pathlib.Path(TESTOVAYA_BAZA).unlink(missing_ok=True)

from fastapi.testclient import TestClient
from main import app

client = TestClient(app)


def test_obrashenie_sohranyaetsya():
    otvet = client.post("/otpravit", data={
        "imya": "Проверочный Пётр",
        "kontakt": "petr@primer.ru",
        "tekst": "Проверочное обращение",
    })
    assert otvet.status_code == 200

    spisok = client.get("/spisok")
    assert "Проверочный Пётр" in spisok.text
```

Первые строки объясняются просто: проверка обязана работать в своей песочнице. Если пустить её в настоящую базу, каждый прогон будет добавлять туда Проверочного Петра, и через неделю в рабочем списке обращений окажется толпа фантомов. Переменная окружения – это настройка, которую программа читает при запуске: удобный способ сказать одному и тому же коду «работай вот с этим файлом», не правя сам код.

Слово `assert` в тесте означает «утверждаю, что это верно». Если утверждение не выполняется, тест падает и показывает, что именно не сошлось. Больше в нём ничего нет: тест – это список утверждений о вашем проекте.

Сам тест читается почти как ваши действия руками: отправить обращение – убедиться, что отправилось – открыть список – убедиться, что обращение в нём есть. Запуск:

```
pytest
```

Зелёная строка с `1 passed` – проверка прошла.

А теперь самое поучительное: сломайте сохранение. Причём сломайте незаметно, как это и происходит в жизни.

Откройте код обработки отправки. Дальше две ветки, потому что агент мог написать работу с базой одним из двух способов – посмотрите, какой у вас.

Если в коде есть отдельная строка со словом `commit` – закомментируйте её. Именно она отдаёт базе команду «сохрани изменения окончательно».

Если такой строки нет, а соединение открыто через `with`, как у нас:

```
    with soedinenie() as db:
        kursor = db.execute("INSERT INTO obrasheniya ...",
(...))
        nomer = kursor.lastrowid
```

то сохранение делает сам `with`, когда блок заканчивается. Отнимите у него эту работу – откройте соединение обычной строкой и закройте вручную, без сохранения:

```
    db = soedinenie()
    kursor = db.execute("INSERT INTO obrasheniya ...", (...))
    nomer = kursor.lastrowid
    db.close()
```

Обратите внимание: строки внутри стали на один уровень левее, потому что блока `with` больше нет. Код рабочий, ошибок в нём нет.

Если сомневаетесь, что получилось то же самое, сверьтесь с готовым проектом по адресу со страницы книги.

Запустите сервис руками, отправьте обращение через браузер. Вы увидите «Обращение принято», номер, никаких ошибок, ничего красного в терминале. Всё как в 0.2 – убедительная картинка. А теперь `pytest`:

```
assert 'Проверочный Пётр' in "<h1>Обращений пока нет</h1>..."
1 failed
```

Проверка поймала то, чего не видно глазами. Обращения не сохранялись, страница как ни в чём не бывало отвечала успехом, ошибок не было ни одной – и без теста вы узнали бы об этом от первого человека, чьё обращение пропало. Верните `with` на место и убедитесь, что снова зелено.

Это и есть ответ на вопрос, зачем нужны автопроверки, если можно щёлкать руками. Руками вы проверяете то, что видно. Тест проверяет то, что должно быть верно.

Одно предостережение, выстраданное. Тест должен проверять суть. Тест «страница открылась и вернула код 200» почти всегда зелёный и почти ничего не значит: страница может открываться, а сохранение быть мертво. Проверяйте то, ради чего сервис существует: обращение дошло до хранилища. Всё остальное – гарнир.

```
git add .
git commit -m "Первый тест: обращение сохраняется"
```

> **Если вы заказчик.** Спросите подрядчика: «Какие проверки выполняются автоматически и что они проверяют?» Ответ «мы всё тестируем руками» означает, что каждая правка проекта может незаметно ломать любое из прежних мест. Это не приговор подрядчику, но это ваша информация о рисках.

## 0.5. Отметка «обработано» и граница первой версии

Сервис принимает и хранит. Для полного маленького цикла не хватает одного: отметить обращение обработанным, иначе список будет только расти и в нём утонет всё.

Задача агенту – по уже знакомому образцу, с данными и границей правки:

```
Добавь к каждому обращению в /spisok кнопку "Обработано".
Нажатие меняет статус обращения на "обработано" в базе.
Обработанные показывай в конце списка, серым.
Меняй только то, что относится к статусу. Форму и сохранение
не трогай.
```

Примите, проверьте руками: нажали – уехало вниз, обновили страницу – состояние держится, перезапустили сервер – держится. Прогоните `pytest`: старый тест обязан остаться зелёным, ведь форму и сохранение мы трогать запрещали. Если он покраснел – агент вышел за границу правки, и вы узнали об этом сейчас, а не через неделю. Это автотест только что отработал свою цену.

Заметьте, как прошёл вечер с точки зрения ритма: одна задача – одна мысль – проверка – коммит. Форма отдельно, хранение отдельно, статус отдельно. Ни разу мы не просили «сделай всё сразу». Этот ритм скучнее, чем «собери мне сервис целиком», и держится он на простом свойстве: большие запросы порождают большие непроверяемые ответы, маленькие проверяются за минуту.

Теперь – граница первой версии. Возьмите файл `PLAN.md` и запишите в него, чего в проекте осознанно нет:

```
Не входит в первую версию:
- вход по паролю и роли
- уведомления на почту
- оформление и стили
- поиск и фильтры по списку
- удаление обращений
```

Список отложенного, существующий на бумаге, отличается от списка в голове одним свойством: он снимает тревогу. Пока «нет авторизации» живёт в голове, оно звучит как упрёк и провоцирует немедленно докручивать. Записанное – превращается в решение: этого нет, потому что так решено, вернёмся, когда дойдёт очередь. Проекты гибнут по обоим сценариям – и когда бросают недоделанными, и когда докручивают без остановки, пока не перестаёт запускаться. Письменная граница защищает от второго.

```
git add .
git commit -m "Статус обработано + граница первой версии"
```

## 0.6. Инструкция для того, кто откроет это через месяц

Остался последний файл, и он адресован конкретному человеку: вам через месяц. Этот человек не помнит ни как запускать проект, ни что в нём работает, ни почему нет авторизации. Проверено на всех, включая авторов: память – негодная система документации, особенно после пары выходных.

Файл называется `README.md`, лежит в корне проекта и отвечает на четыре вопроса. Что это. Как запустить. Что работает. Чего нет.

```
# Приём обращений

Учебный сервис: принимает обращения через форму,
хранит в SQLite, показывает список со статусами.

## Запуск
python3 -m venv .venv                          # только в первый
раз
source .venv/bin/activate                      # Windows:
.venv\Scripts\activate
pip install fastapi uvicorn python-multipart pytest httpx2
uvicorn main:app --reload
Открыть http://127.0.0.1:8000

## Проверка
pytest                                         # работает в
test_zayavki.db, рабочую базу не трогает
Руками: отправить обращение -> увидеть в /spisok ->
перезапустить сервер -> обращение на месте.

## Работает
Форма, сохранение в zayavki.db, список, статус "обработано".

## Не входит в первую версию
См. PLAN.md
```

Пишется за пять минут, окупается в первый же день возвращения к проекту. Есть и второй адресат: любой человек, которому вы проект передадите – помощнику, подрядчику, покупателю. Инструкция запуска, работающая на чужой машине, – это признак того, что проект существует отдельно от вашего компьютера. До этого момента он существовал только вместе с вами.

```
git add .
git commit -m "README: запуск, проверка, границы"
```

> **Если вы заказчик.** То, что вы сейчас видели, – зерно комплекта передачи: минимального набора, который должен оставаться у вас после любой сданной работы. Из чего он состоит целиком и как проверить, что вам его действительно от-

дали, – разбор в части IV. Требовать его можно уже сейчас, любыми словами.

## 0.7. Что произошло за вечер

Посмотрим на результат. В папке `zayavki` лежит сервис, который принимает обращения, хранит их, переживает перезапуск, показывает список и умеет отмечать сделанное. Рядом – история изменений с рабочими точками, одна автоматическая проверка сути, письменная граница версии и инструкция, по которой сервис запустит другой человек. По меркам промышленной разработки – зерно. Но это зерно, у которого есть ответы на все шесть вопросов из 0.2.

И у вас было главное событие вечера: форма, которая говорила «Спасибо» и выбрасывала обращения. Вы видели, как убедительно выглядит результат, у которого внутри пусто, – и как быстро это вскрывается, если знать, куда смотреть. Всё дальнейшее в книге выросло из этого зазора. Генерация даёт правдоподобное; проверка отличает правдоподобное от работающего; эксплуатация – искусство не дать работающему тихо испортиться. Агенты сегодня закрывают первое звено блестяще, второе – кое-как, третье – никак. Значит, второе и третье – ваша работа, и теперь вы знаете, из чего она состоит.

И оговорка на дорогу. Стек этого вечера – учебный: мы выбрали его за то, что он ставится одной командой и одинаково работает у всех. Если ваш проект живёт на другом языке или собран в другом инструменте – ничего из дальнейшего это не отменяет: правила книги про работу и владение, к конкретному стеку они не привязаны.

В части I мы займёмся ремеслом: как описывать задачу, чтобы агент реже заполнял пустоты догадками, как читать то, что он вер-

нул, как держать его в границах – а ещё как устроена его память, что делать, когда окон с агентами два, и чем мерить размер затеи. Приёмы, которые сегодня работали по нашей подсказке, там станут системой. Читать часть I лучше по одной-две главы за вечер, применяя каждую к своему сервису, – торопиться некуда, книга не убежит. Приёмы, которые вы применяли сегодня по нашей подсказке, там станут системой.

Сервис не выключайте. Он нам ещё пригодится.

# Часть I. Ремесло

## Постановка задачи

В первый вечер вы ставили агенту задачи по нашим подсказкам, и они срабатывали. Пришло время разобрать, почему они были устроены именно так, – и научиться собирать такие самому, без подсказок, под любую свою затею.

Начнём с фразы, которую стоит повесить над рабочим местом: агент делает не то, что вы имели в виду. Агент делает то, что вы описали. Между «имел в виду» и «описал» всегда есть зазор, и всё, что в этот зазор попало, агент заполнит сам – быстро, уверенно и правдоподобно. Форма из первого вечера, выбрасывавшая обращения, родилась ровно в таком зазоре: про сохранение в задаче не было ни слова, и агент выбрал самый дешёвый вариант.

Отсюда определение ремесла, которому посвящена эта часть. Постановка задачи – это работа по сужению зазора. Не поиск волшебных слов, не вежливость и не длина: техника. У неё есть анатомия, и её можно освоить за одну главу.

### Анатомия: от одной строки до рабочей задачи

Возьмём пример из вашего же проекта. У сервиса из части 0 копятся обращения, и листать список стало неудобно. Естественное желание – поиск. Он, кстати, лежит у вас в списке отложенного – и пусть лежит: здесь мы разбираем постановку, строить не обязательно. Естественная формулировка:

```
Добавь поиск по обращениям.
```

Поставьте себя на место исполнителя, который получил эту строку и не может переспросить. Искать по каким полям – по имени, по

тексту, по всему сразу? Где искать – на странице списка или отдельной страницей? Что показывать, когда ничего не нашлось? А когда строка поиска пуста? Трогать ли существующий список или делать рядом? Каждый из этих вопросов агент решит за вас, не сообщая, что решал. Получите вы сумму чужих догадок, оформленную как готовый результат, – и будете разбираться с ней дольше, чем писали бы задачу.

Теперь соберём задачу по частям, добавляя по одному элементу и глядя, что меняется.

**Первое: что нужно, в терминах результата.**

```
Добавь на страницу /spisok поиск по обращениям:
одно поле ввода, поиск по имени и тексту обращения.
```

Уже лучше: определено, где живёт поиск и по каким полям идёт. Но по-прежнему не сказано, как понять, что готово, – и не сказано, чего не делать.

**Второе: граница правки.** Вы знаете этот приём с первого вечера, теперь у него есть имя и постоянное место – последней строкой любой задачи:

```
Меняй только страницу списка. Форму, сохранение и базу не
трогай.
```

Цену этой строки вы уже заплатили в первый вечер, когда ставили её впервые: без неё правка расползается по проекту.

**Третье: чем проверять.** Самый недооценённый элемент. Опишите проверку – и вы автоматически описали поведение во всех важных случаях:

```
Проверка: по слову из текста обращения находится нужное;
по пустому полю показывается весь список;
по слову, которого нет нигде, – сообщение «ничего не найдено»
и ссылка на полный список.
```

Заметьте, что произошло: формулируя проверку, вы приняли три решения о поведении – те самые, которые иначе принял бы агент. Пустое поле, пустой результат, путь назад. Это главный секрет элемента «чем проверять»: он заставляет вас продумать края задачи до того, как код существует. Края – то место, где живут почти все поломки.

**Четвёртое: чего не делать.**

```
Не добавляй фильтры, сортировку и постраничную разбивку – не
сейчас.
```

Без запретов агент достраивает «полезное»: раз поиск, то и фильтры, и сортировка, и разбивка по страницам. Каждая добавка увеличивает объём непроверенного. Запрет – это способ получить ровно один проверяемый шаг вместо трёх непроверяемых.

Соберём целиком:

```
Добавь на страницу /spisok поиск по обращениям:
одно поле ввода, поиск по имени и тексту обращения.
Проверка: по слову из текста находится нужное обращение;
по пустому полю показывается весь список; по слову, которого
нет нигде, – сообщение «ничего не найдено» и ссылка на полный
список.
Не добавляй фильтры, сортировку и постраничную разбивку.
Меняй только страницу списка. Форму, сохранение и базу не
трогай.
```

Восемь строк вместо одной. Время на написание – три минуты. Эти три минуты вы окупите в первые же десять: результат либо совпадёт с описанным, либо расхождение будет видно сразу, потому что

есть с чем сравнивать. Задача из одной строки не оставляет даже возможности заметить расхождение – сравнивать не с чем.

Четыре элемента стоит запомнить как формулу: **что нужно, чем проверять, чего не делать, где граница.** Всё остальное в задаче – украшение. Длинные вступления, вежливые обороты и описания настроения на результат не влияют; влияют эти четыре.

## План до кода

Для задач побольше – новая страница, изменение хранения, всё, что затрагивает несколько мест, – к формуле добавляется ход, который бережёт больше всего нервов: попросить план до исполнения.

```
Пока не пиши код. Опиши, что собираешься сделать: какие файлы
изменишь, какие добавишь, как будет храниться и проверяться.
```

Агент отвечает текстом на полстраницы, и этот текст – самая дешёвая точка контроля во всём цикле. Ошибка, пойманная в плане, стоит одну реплику: «нет, храни в той же базе, отдельный файл не нужен». Та же ошибка, пойманная в готовом коде, стоит разбора, переделки и нового прогона проверок. Непонимание задачи видно в плане целиком – в коде оно размазано по файлам.

У этого хода есть отраслевое имя – разработка от спецификации: сначала согласованное описание, потом исполнение, потом сверка результата с описанием. Существуют инструменты, которые строят вокруг этого весь рабочий цикл. Знать об этом полезно по одной причине: приём придуман не нами и хитростью не является – это нормальный способ работать, к которому индустрия пришла на тех же ожогах.

И парный ход, ещё дешевле, – передать вопросы на сторону агента:

```
Прежде чем начать, задай мне вопросы, без которых задача
получится слишком общей.
```

Это работает лучше, чем кажется, по свойству самой модели: она хорошо видит пробелы в чужом тексте, хуже – в собственных планах. Три-четыре вопроса из ответа обычно попадают в те самые края, которые вы не продумали. Отвечать на вопросы легче, чем предугадывать их, – пусть спрашивает.

## Антипример: задача-мешок

Для полноты – как выглядит противоположность. Реальный жанр, встречается постоянно:

```
Сделай нормальный поиск по обращениям, и заодно поправь дизайн
списка,
а то он страшный, и ещё уведомления бы прикрутить, чтобы на
почту
приходило, и посмотри, почему вчера медленно работало.
```

Четыре задачи в одной, ни у одной нет ни проверки, ни границы, а «заодно» и «ещё бы» открывают агенту дверь в любую часть проекта. Результат предсказуем: большая правка во многих файлах, где что-то из четырёх сделано, что-то нет, что-то сделано не так, – и непонятно, с какого конца проверять. Слово «заодно» в задаче агенту вообще стоит запретить себе. К нему мы ещё вернёмся в части про изменения работающего проекта, но рождается оно именно здесь, при постановке.

Правило деления простое. Если в задаче есть «и ещё» – это две задачи. Если результат нельзя проверить одним коротким сценарием – задача велика, делите. Маленькие задачи кажутся медленным путём; на дистанции они быстрее, потому что не порождают разборов «что из этого вообще работает».

## Библиотека формулировок

Шесть заготовок, которые закрывают большинство ситуаций. Они уже встречались вам по одной – вот все вместе, в том виде, в каком

их стоит держать под рукой:

**Граница правки.** «Меняй только то, что относится к [задаче]. [Перечень] не трогай». Последней строкой каждой задачи.

**План до кода.** «Пока не пиши код. Опиши, какие файлы изменишь и добавишь, как будет храниться и проверяться». Для всего, что больше одного файла.

**Вопросы к задаче.** «Прежде чем начать, задай мне вопросы, без которых задача получится слишком общей». Когда сами чувствуете, что описали смутно.

**Критерий готовности.** «Готово, когда: [проверяемый сценарий]». Заставляет продумать края.

**Запрет на переделку.** «Не переписывай работающее. Если считаешь, что нужна переделка, – сначала объясни, что и зачем, не делая». Против «улучшений» по собственной инициативе.

**Требование показать проверку.** «Покажи, как ты проверил, что это работает. Что запускал, что получил». Приучает и агента, и вас, что «сделал» без «проверил» не считается.

Замечание к библиотеке. Ценность этих формулировок не в словах – слова можете менять как угодно. Ценность в том, какие решения они у вас требуют: назвать границу, описать проверку, отделить нужное от попутного. Это решения владельца, и делегировать их некому: агент исполняет, решает тот, кто ставит задачу.

> **Если вы заказчик.** Формула «что нужно, чем проверять, чего не делать, где граница» – это же скелет хорошего задания подрядчику. Если в вашем задании есть только первый элемент, три остальных подрядчик решит за вас – ровно как агент, только дороже и дольше. Особенно окупается «чем проверять»: задание с проверяемым критерием готовности лишает смысла спор «а мы считаем, что готово».

> **Упражнение.** Возьмите последнюю задачу, которую вы давали агенту, – настоящую, из своей переписки. Разберите её по четырём элементам и найдите отсутствующие; в типичной задаче из жизни присутствует один элемент из четырёх. Перепишите по формуле и дайте агенту заново, в свежей сессии. Сравните два результата – это сравнение убеждает лучше любой главы.

Задача поставлена – агент вернул результат. Следующая глава про вторую половину цикла, о которой почти никто не говорит: как читать то, что вернулось, не читая кода целиком, и как отличать отчёт о работе от самой работы.

## Как читать то, что вернул агент

Задача поставлена по всем правилам прошлой главы, агент поработал и вернул результат: пачку файлов, правки в существующих и бодрый отчёт о проделанном. Наступает момент, который новичка пугает больше всего: это надо как-то оценить. Кода много, понимаете вы в нём не всё, а нажать «принять» не глядя – значит впустить в проект неизвестно что.

Хорошая новость: читать код целиком вам не нужно. Плохая: совсем не читать нельзя. Между этими крайностями лежит навык выборочного чтения: вместо понимания каждой строки вы ищете в результате ответы на короткий список вопросов. Этому навыку и посвящена глава.

Сразу договоримся о цели. Цель чтения – ответ на один вопрос: понял ли исполнитель задачу. Непонимание видно сверху, без глубокого погружения, – как по расстановке мебели видно, что грузчики перепутали квартиры, ещё до проверки каждого ящика.

### Чтение по слоям

Порядок фиксированный, четыре слоя, от поверхности вглубь. Он устроен так потому, что первые поломки почти всегда живут в одном из этих слоёв, а не в глубине алгоритмов.

**Слой первый: что появилось.** Ещё не открывая ни одного файла, посмотрите на их список. Сколько новых, сколько изменённых, совпадает ли это с ожиданием. Вы просили поиск на существующей странице – а в результате шесть новых файлов и какая-то новая папка? Стоп. Разговор здесь ещё даже не о качестве кода – о несовпадении масштаба: исполнитель понял задачу шире, чем вы ставили. Масштаб результата – самый быстрый индикатор понимания, и читается он за десять секунд.

**Слой второй: где данные.** Вы уже знаете эту привычку из первого вечера, теперь она становится пунктом порядка. Что происходит с данными: появилось ли новое хранение, изменилась ли структура существующего, куда пишется и откуда читается. Правки в слое данных – самые дорогие в исправлении: код переписывается, а данные, записанные в неудачную структуру, приходится переносить. Если задача не предполагала изменений хранения, а они есть, – выяснять, зачем, раньше всего остального.

**Слой третий: границы наружу.** Всё, чем проект касается внешнего мира: адреса страниц, обращения к другим сервисам, письма, файлы на диске. Новая граница – это новое место, где проект может сломаться о реальность, и новое место, которое надо проверять. Появилась отправка почты, которой вы не просили? Обращение к внешнему сервису «для удобства»? Каждая такая граница должна быть или заказана вами, или объяснена, – третьего варианта нет.

**Слой четвёртый: что при ошибке.** Это третий вопрос первого вечера, дословно. Найдите глазами, что происходит, когда что-то идёт не так: пустой ввод, отсутствие записи, недоступная база. Не

вникая в механику – просто убедитесь, что эти случаи в коде вообще существуют. Код, в котором предусмотрен только удачный путь, – это та самая картинка из первого вечера, только на уровень глубже: работает, пока всё хорошо.

Четыре слоя занимают десять-пятнадцать минут и не требуют читать алгоритмы. По их итогам у вас есть обоснованный ответ на главный вопрос – понял или не понял, – и список мест, куда смотреть пристальнее.

## Объясни как маршрут

Теперь приём, который превращает агента из экзаменуемого в экскурсовода – и одновременно проверяет его работу лучше любого допроса:

```
Проведи меня по маршруту одного действия: человек заполнил
форму
и нажал «Отправить». Что происходит дальше, шаг за шагом,
с указанием файла на каждом шаге?
```

Ответ – связный рассказ: браузер отправляет данные туда-то, их принимает такая-то функция в таком-то файле, проверяет то-то, пишет туда-то, отвечает тем-то. Ваша работа – идти по рассказу с открытыми файлами и сверять: этот файл существует? эта функция в нём есть? она действительно пишет туда, куда сказано?

Маршрут хорош тем, что линеен: его проверяют шаг за шагом, не держа в голове всю систему. И он мгновенно вскрывает разрыв между рассказом и кодом – а этому разрыву посвящён следующий раздел, самый важный в главе.

## Отчёт – это гипотеза

Правило, ради которого стоило писать эту главу, мы сформулировали для себя после конкретного опыта, и расскажем его как есть.

Мы регулярно отдаём код на разбор моделям – свежий взгляд находит то, что замылилось. И в этих разборах, среди настоящих находок, стабильно попадаются два вида брака. Ложные тревоги: модель уверенно описывает поломку, которой нет, – например, потому что в постановке разбора была неточная вводная, и модель на неё добросовестно опёрлась. И выдумки: модель ссылается на строку, которой в файле не существует, или цитирует код, которого никто не писал, – уверенно, с номерами строк, с выводами. По нашим журналам это не редкость: в одном из разборов часть находок оказалась ложной, в другом модель забыла половину того паттерна, который бралась проверять.

С тех пор у нас действует правило без исключений: **находка модели – это гипотеза. Документом является только сам код и результат запуска.** Каждую находку, прежде чем чинить, мы сверяем с настоящим файлом и настоящим поведением. Чинить вслепую по отчёту запрещено – так можно «исправить» работающее. Дважды мы объявляли поломкой то, что работало исправно, и подробный разбор этих двух промахов ждёт в части про эксплуатацию.

Для вас это правило звучит так. Всё, что агент говорит о своей работе, – «я добавил проверку», «данные сохраняются», «этот случай обработан» – требует предъявления. Не потому что агент лжёт: он добросовестно описывает то, что собирался сделать, и его рассказ о намерении неотличим по тону от рассказа о сделанном. Разницу между намерением и фактом устанавливаете вы – взглядом в файл, запуском, тестом.

Практическое следствие для маршрута из прошлого раздела: расхождение между рассказом и кодом – находка, а не придирка. Если агент описывает шаг, которого в коде нет, вы поймали ровно тот зазор, из которого потом вырастают «странно, должно же работать».

## Признаки непонимания

Короткий определитель. Пять признаков, каждый виден без чтения алгоритмов, каждый означает, что задача понята не так, – и чем раньше вы это признаете, тем дешевле выйдет.

Слишком много нового. Просили правку – получили россыпь файлов. Масштаб выдаёт додумывание.

Переименовано без просьбы. Ваши имена – из задачи, из файла правил, из прошлых шагов – заменены на «более правильные». Значит, агент работал от своих привычек, ваш проект он в этот момент не слышал.

Новые зависимости. В проекте появились сторонние библиотеки, которых никто не заказывал. Каждая – чужой код в вашем доме, и решение о вселении принимали не вы.

«На будущее». Заготовки, пустые функции, настройки «на вырост». Это объём без пользы: проверять нечего, а поддерживать придётся.

Ответ шире вопроса. Спрашивали про одно место – переделано три. Знакомо по прошлой главе: границы в задаче не было или она не удержала.

## Что делать с плохим результатом

Последний раздел – о реакции, и он короткий, потому что правильная реакция одна.

Не спорить. Разговор в духе «ну я же просил по-другому, почему ты...» – потраченное время и захламлённый разговор: вы обсуждаете с исполнителем его прошлую ошибку вместо будущего результата. Модель не обучается от вашего недовольства внутри сессии – она просто получает ещё текста.

Вместо спора – сузить и переспросить. Взять свою же задачу, найти, где она позволила неверное толкование (обычно место находится – прошлая глава давала формулу, по которой искать), дописать границу или проверку и дать заново. Часто – в свежей сессии, чтобы неудачная попытка не тянулась следом. Переписанная задача плюс новый заход почти всегда дешевле, чем лечение неудачного результата серией уточнений: после трёх «нет, я имел в виду...» в разговоре лежат три слоя противоречащих указаний, и результат портится дальше.

И граница этого правила, чтобы оно не превратилось в привычку всё выбрасывать: если результат верен по сути и плох в деталях – детали правятся точечными задачами, это нормальная работа. Выбрасывать и переспрашивать стоит тогда, когда неверно понята сама задача.

> **Если вы заказчик.** Приём «проведи меня по маршруту» работает и с подрядчиком, дословно: «расскажите путь одной заявки от кнопки до менеджера, с названиями систем на каждом шаге». Знание кода не требуется – требуется готовность заметить, где рассказ становится туманным. Шаг, на котором звучит «ну там дальше это обрабатывается», – это то место, которое стоит попросить показать. И правило про гипотезу здесь то же: отчёт о работе и работа – разные вещи, предъявление разницы не обидит никого, кому есть что предъявить.

> **Упражнение.** Откройте свой вечерний проект и попросите агента: «Объясни, что делает файл main.py и зачем в нём каждая функция». Затем проверьте рассказ по файлу, строка за строкой. Ищите два вида расхождений: сказано о том, чего в файле нет, и пропущено то, что есть. Найдёте хотя бы одно – вы своими глазами увидели зазор между отчётом и фактом на файле в сто строк. Помните об этом зазоре каждый раз, когда

> услышите уверенное «всё готово» о проекте в десять тысяч строк.

Вы умеете ставить задачу и читать результат. Но некоторые вещи приходится повторять агенту каждый раз: не трогай это, проверяй вот так, называй по-нашему. В следующей главе мы переселим эти повторения из вашей головы в проект – в файл правил, который агент читает сам. Заодно выяснится, что лучшие правила пишутся после ожогов, никогда заранее, и что наш собственный файл правил – это, по сути, дневник наших аварий.

## Файл правил проекта

К этому моменту у вас накопились вещи, которые вы говорите агенту постоянно. Не трогай форму и сохранение. После правки прогони тест. Имена полей – как в проекте, не переименовывай. Каждая новая сессия начинается с того, что вы повторяете всё это заново, потому что агент, как мы выясним подробнее в следующей главе, между сессиями не помнит ничего.

Правило этой главы простое: всё, что вы объяснили агенту дважды, должно быть записано в проекте. В третий раз объяснять должен файл.

### Что это такое

Файл правил – обычный текстовый файл в корне проекта, который агент читает перед началом работы и держит перед глазами всю сессию. Вы пишете туда указания один раз – агент подчиняется им в каждой сессии, не спрашивая и не забывая.

У этого файла есть устоявшееся отраслевое имя – AGENTS.md. Стоит знать, что это не особенность какого-то одного продукта: формат стал общим стандартом, передан в независимый фонд и читается

парой десятков инструментов. Практический смысл для вас: правила, записанные однажды, переживут смену инструмента – новый агент прочтёт тот же файл.

Что туда пишут – по опыту, шесть разделов, все короткие:

Как запускать и как проверять. Команды запуска, команда тестов, что считается зелёным.

Что запрещено трогать. Файлы и области, которые меняются только по явной просьбе.

Обязательные действия. Что агент делает всегда, без напоминания: прогнать тесты после правки, зафиксировать рабочую точку после зелёного.

Соглашения об именах. Как называются поля, страницы, файлы – чтобы «более правильные» имена из привычек модели не расползались по проекту.

Стиль текстов, если проект что-то пишет людям: тон сообщений, язык, чего в текстах не бывает.

Куда что кладётся. Где живут страницы, где логика, где проверки – карта проекта в пять строк.

Для вечернего сервиса из части о файл правил выглядит так – целиком, без сокращений:

```
# Правила проекта

## Запуск и проверка
Запуск: uvicorn main:app --reload
Тесты: pytest. После любой правки кода - прогнать.
Зелёные тесты обязательны перед коммитом.

## Не трогать без явной просьбы
Форму отправки, сохранение в базу, файл test_zayavki.py.

## Имена
Поля формы: imya, kontakt, tekst. Не переименовывать.
Страницы: / (форма), /spisok (список). Новые - согласовать.

## Всегда
Одна задача - одна правка. За границу задачи не выходить.
Если нужна переделка работающего - сначала объяснить, не
делая.
```

Семнадцать строк. Каждая сессия теперь начинается с этого файла, ваши повторения больше не нужны – и заметьте, что половина строк вам знакома: это библиотека формулировок из главы про постановку, переехавшая на постоянное место жительства.

## Осадок ожогов

Теперь главная мысль главы, и она про то, откуда берутся правила, которые работают.

Наш собственный файл правил вырос до нескольких страниц. Читать его постороннему скучно: запреты, пороги, перечни, никакой философии. Но у почти каждой строки там есть дата рождения, и это дата аварии.

Строка «этот сайт собирается только своим сборщиком, чужим – запрещено» появилась после того, как два сборщика, писавшие в один файл стилей, оставили сто тридцать шесть статей без оформления; подробно этот случай мы разберём в части про изменения работающего проекта. Строка «не переименовывать поля – на име-

нах держатся проверки» стоит там по той же причине, по которой вы сами задавали имена в задаче агенту: однажды переименованное поле обрушило проверки, и полчаса ушло на выяснение, почему красное то, что никто не трогал. Строка «проверять перезапуском, а не обновлением страницы» – после того как данные, «которые точно сохранялись», не пережили остановку службы. Ожог у каждой строки свой, и в файле он записан рядом одной фразой.

Мы называем это про себя «осадок ожогов», и в названии есть точность: правила не сочинялись – они выпали в осадок из настоящих потерь. Отсюда свойство, которое отличает их от правил из учебника: они соблюдаются. Правило, написанное «на всякий случай», не соблюдает никто, включая автора, – оно ничем не подкреплено, и рука обходит его без угрызений. Правило, написанное после ожога, помнит свою цену, и цена записана рядом одной фразой: нарушишь – получишь вот это, проверено тогда-то.

Отсюда же ответ на вопрос, который возникает у каждого, кто впервые слышит про файл правил: а что туда писать сначала? Ничего. Пустой файл с заголовком – нормальный старт. Правила придут сами, и глава подсказывает, откуда: первое же «я ведь это уже объяснял» – кандидат в файл. Заметили, что повторяетесь, – остановились, записали, продолжили. Файл растёт со скоростью ваших ожогов, и это правильная скорость: каждая строка в нём заработана.

Как записывать – три требования. В повелительном наклонении: «не трогать», «прогнать», «согласовать» – агент лучше слушается прямых форм. Коротко: правило длиной в абзац не прочтёт никто. И с причиной одной фразой: «не переименовывать поля – на именах держатся тесты». Причина превращает запрет из каприза в довод, а заодно напоминает вам самим через полгода, зачем это здесь.

## Чего в файле не должно быть

Файл правил портится двумя способами, оба от усердия.

Первый – пересказ очевидного. Пожелания вроде «пиши чистый код», «делай хорошо», «следуй лучшим практикам» не значат ничего: агент и так старается по своим представлениям, а ваши представления в этих словах не выражены. Туда же – пересказ документации инструментов: агент её знает лучше вас. Каждая пустая строка в файле разбавляет полные: чем больше воды, тем меньше веса у настоящих запретов.

Второй – правила впрок, про ситуации, которых у вас не было. Они кажутся предусмотрительностью, а работают как шум: вы сами не помните, почему это здесь, и при первом неудобстве отменяете. Файл правил – не свод законов идеального проекта. Это память вашего конкретного проекта о его конкретных ошибках.

Проверка качества файла простая: уберите правило и посмотрите, станет ли больно. Правило, отмена которого ничего не меняет, было балластом.

## Файл правил как имущество

И последнее, короткое. Взгляните на файл правил глазами части IV – как на предмет комплекта владельца.

README объясняет, как запустить. Список ограничений – что не сделано. Файл правил закрывает третий вопрос, который встанет у любого преемника: как здесь принято работать и на чём здесь уже обжигались. Человек, получивший проект с настоящим файлом правил, получает не только код – он получает прививки от аварий, которые ему теперь не обязательно повторять. В нашей ежедневной передаче между сессиями этот файл делает ровно эту работу: новая сессия не помнит вчерашних ошибок, но подчиняется правилам, которые из них выросли. Память стирается – осадок остаётся.

> **Если вы заказчик.** Спросите подрядчика, есть ли в проекте файл правил для агентных инструментов, и попросите показать. Сам факт его существования говорит о зрелости процесса больше, чем портфолио: значит, здесь работают с агентами системно и записывают выводы из ошибок. А его содержимое – это, по сути, дневник того, на чём проект уже обжигался. Полезнейшее чтение перед тем, как принять работу.

> **Упражнение.** Создайте в вечернем проекте файл правил из трёх заработанных строк, выдуманные не годятся: вспомните, что вы уже дважды говорили агенту за время этой книги, и запишите ровно это. Затем проверьте делом: начните свежую сессию, дайте задачу, которая раньше требовала напоминаний, – и посмотрите, соблюдены ли правила без единого слова с вашей стороны. Это упражнение занимает десять минут и меняет характер работы: с этого дня ваши объяснения накапливаются, вместо того чтобы испаряться.

Файл правил решает проблему повторения между сессиями. Но у беспамятства агента есть и второй этаж: внутри одной сессии его внимание тоже конечно, расходуется и к вечеру тупеет. Следующая глава – про контекст, память и стоимость: что агент на самом деле «видит», почему длинный разговор портится и как распределять работу между моделями, чтобы платить за силу только там, где она нужна. Заодно расскажем, как мы сами делим работу между моделями.

## Контекст, память и стоимость

Прошлая глава несколько раз повторила «агент между сессиями не помнит ничего» и обещала объяснить. Объясняем – потому что из устройства памяти агента следует половина странностей, с которы-

ми вы уже столкнулись, и почти все способы работать дешевле и лучше.

Понадобится одна картинка, без техники. У агента есть рабочий стол ограниченного размера – он называется контекстом. На столе лежит всё, что агент сейчас учитывает: ваша задача, файл правил, куски кода, которые он открыл, история текущего разговора, его собственные промежуточные рассуждения. Что на столе – то существует. Чего на столе нет – того для агента не существует вовсе: ни вчерашней сессии, ни соседней папки, ни ваших намерений, ни этой книги.

Вся глава – три следствия из этой картинки.

## Следствие первое: длинный разговор тупеет

Стол конечен. Пока разговор короткий, на столе лежит нужное и только нужное: задача, правила, свежие файлы. К третьему часу работы там громоздится всё подряд: пять прошлых задач, три неудачные попытки, куски кода, который вы уже переписали, ваши уточнения, отменяющие друг друга. Новое кладётся поверх, старое уминается и теряет подробности – и агент начинает вести себя знакомым каждому образом: путает свежую версию со старой, «вспоминает» уже отменённые решения, отвечает всё более общо.

Это не поломка и не усталость в человеческом смысле. Это переполненный стол: среди наваленного труднее найти относящееся к делу, а утрамбованное старое теряет точность. Важное следствие для вашей интуиции: качество работы агента зависит от того, что лежит на столе, сильнее, чем от любой формулировки задачи. Блестящая постановка посреди захламлённой сессии проигрывает средней постановке в чистой.

Отсюда два рабочих правила.

Свежая сессия – бесплатное лекарство. Закончили задачу – зафиксировали результат – закрыли разговор. Новую задачу начинайте новым разговором: на столе окажутся файл правил, код в его текущем состоянии и одна задача. Всё нужное, ничего лишнего. Держать один бесконечный разговор «потому что он уже в курсе» – ложная экономия: «в курсе» на переполненном столе означает «путает».

Ссылка лучше пересказа. Не рассказывайте агенту, что в файле, – скажите, какой файл посмотреть: он положит на стол настоящий текст, а не ваш пересказ по памяти, который может расходиться с действительностью. Ровно из этого же правила растёт привычка предыдущей главы: правила живут в файле, а не в ваших повторениях, – файл всегда точен, память всегда примерно.

### Следствие второе: между сессиями памяти нет

Закрыли разговор – стол очищен. Полностью. Следующая сессия начинается с пустого стола, и никакие «помнишь, мы вчера...» не работают: не помнит. Вчерашней сессии больше не существует.

Первая реакция на это – досада: как работать с исполнителем, который каждый день всё забывает? Но вы уже знаете ответ по главе о файле правил. Беспамятство лечится имуществом проекта, память тут и не нужна: файл правил помнит, как здесь принято работать; README – как запускать; список ограничений – что не сделано; история изменений – что и когда менялось. Сессия, начавшая с пустого стола, кладёт на него эти четыре вещи и через минуту знает о проекте всё, что нужно для работы.

У нас проект каждый день переходит к сессии, которая не помнит ничего, и это оказалось той же задачей, что передача проекта человеку, – к ней мы вернёмся в последней части. Здесь важна обратная сторона: всё, что вы делаете для агента – правила, README, журнал решений, – автоматически оказывается сделанным для любого

будущего человека. Дисциплина работы с беспамятным исполнителем и дисциплина передачи – одна и та же дисциплина. Вы получаете вторую бесплатно.

## Следствие третье: сила стоит денег, и её надо тратить прицельно

Агентная работа платная – по подписке или за объём, неважно: важно, что сильная модель дороже слабой, а длинная сессия дороже короткой. Цен и названий мы по правилам этой книги не называем – они устареют раньше тиража. Называем принципы, они держатся годами.

Главный принцип: сила модели должна соответствовать задаче, а не вашей тревоге. Тревога подсказывает всегда брать самую сильную – «чтобы наверняка». На деле у большинства задач вечернего проекта есть уровень, выше которого разницы нет: переименовать, добавить поле, написать простую страницу одинаково сделает и средняя модель, и великая. А есть задачи, где разница драматична: спроектировать структуру, распутать поломку, вычитать текст на повторы.

Наше рабочее правило распределения, целиком: механическую работу – дешёвой модели; тонкую, где важен вкус или глубина, – дорогой; проверку сделанного – другой модели, чем та, что делала. Последняя часть неочевидна и важнее двух первых: исполнитель плохо проверяет сам себя, и дело здесь в положении, сила ни при чём. Тот, кто написал, смотрит на текст изнутри своего замысла и видит замысел; свежий взгляд видит текст.

Так мы работаем сами: структуру задач и разборов поручаем одной модели, тонкую работу со словом – другой, а проверку сделанного всегда третьей, со своим чистым столом. Ни одна из этих ролей не

взаимозаменима с другой, и общий счёт при таком разделении ниже, чем если бы всё делала самая дорогая модель.

К принципу прилагается правило эскалации, выстраданное: если задача не поддалась с двух заходов – дело уже вряд ли в формулировке. Третья попытка «сказать то же самое другими словами» стоит денег и обычно не помогает. Помогают два хода: взять модель сильнее – или разрезать задачу на части, каждая из которых по зубам текущей. Часто второе лучше первого: задача, которая не даётся, обычно велика, и это лечится делением; по-настоящему сложные встречаются реже, чем кажется.

> **Если вы заказчик.** Из этой главы следует вопрос к смете подрядчика, работающего с агентами: «как у вас распределены модели по задачам?» Ответ «мы всё делаем самой мощной» означает, что вы оплачиваете стрельбу из пушки по воробьям; ответ с распределением – признак считающего процесса. Второй вопрос: «кто проверяет сделанное – та же модель или другая?» Теперь вы знаете, почему это важно.

> **Упражнение.** Проведите опыт на собственном агенте. Возьмите один и тот же вопрос средней сложности о вашем проекте – например, «объясни, как устроено хранение обращений и что произойдёт при двух одновременных отправках». Задайте его дважды: первым сообщением в свежей сессии – и тем же текстом в конце длинной рабочей сессии, где вы час правили код. Сравните два ответа: полноту, точность, конкретность. Разница, которую вы увидите, – это цена захламлённого стола, и после этого опыта закрывать сессии станет легко.

Остался последний источник хаоса, о котором мы пока говорили вскользь: вторая сессия. Стоит открыть два окна – и у вашего проекта уже двое исполнителей, которые не знают друг о друге, а общие файлы знают обоих. Следующая глава – про работу несколь-

ких рук на одном проекте: почему это наступает раньше, чем вы будете готовы, и какая дисциплина спасает.

## Несколько рук на одном проекте

Эта глава могла бы стоять в конце книги, в разделе для продвинутых, – так обычно и делают, потому что «работа в команде» звучит как забота больших проектов. Мы ставим её в первую часть, и на то есть причина, проверенная на себе: несколько рук на проекте появляются раньше, чем вы успеваете подготовиться. Для этого не надо никого нанимать. Достаточно открыть второе окно.

Выглядит это невинно. В одном окне агент делает долгую задачу – скажем, переделывает страницу списка. Вам скучно ждать, и во втором окне вы просите другого агента поправить мелочь – текст сообщения, цвет кнопки. Поздравляем: у вашего проекта теперь двое исполнителей. Они не знают друг о друге. Проверять друг друга они не умеют. А файлы у них общие.

Дальше происходит то, что в командах происходило десятилетиями, только быстрее. Оба читают файл в одном состоянии, оба правят своё, оба сохраняют – и чья запись легла второй, того и файл. Работа первого исчезает без звука: ни ошибки, ни предупреждения, просто в файле её больше нет. Вы узнаете об этом позже – когда «уже же починенное» окажется сломанным, и никто не сможет объяснить, как.

У нас эта механика встречается в журналах не раз, и подробный разбор ждёт в части про изменения работающего проекта. Здесь достаточно короткого: параллельная сессия способна похоронить чужую работу молча, и особенно коварен отложенный вариант – когда правка на месте в момент проверки и исчезает позже, потому что вторая сессия сохранила поверх неё своё, взяв за основу устаревшую копию.

Из-за скорости агентов старая командная проблема стала злее: человек правит файл минутами и часами, агент – секундами, окна конфликтов стали чаще, а привычки защищаться у одиночки нет. Вырабатываем.

## Дисциплина: три правила

**Одна задача – одно окно – своя территория.** Главное правило, оно предотвращает большинство конфликтов до их рождения. Двум одновременным сессиям нельзя давать задачи с пересекающейся областью правки. Границы, которые вы и так пишете в каждой задаче с главы о постановке, здесь получают вторую работу: это не только забор для агента, но и раздел территории между агентами. «Меняй только страницу списка» в одном окне и «меняй только тексты сообщений» в другом – два непересекающихся участка; конфликтовать негде. Если же две задачи тянутся к одному файлу – не запускайте их одновременно, сделайте по очереди. Очередь медленнее на минуты и дешевле на часы.

**Фиксировать чаще, чем кажется нужным.** Коммит – ваша единственная защита от молчаливой потери: затёртое в файле восстанавливается из истории за минуту, если оно в историю попало. Правка, не попавшая в коммит, не защищена ничем, и при параллельной работе это утраивается. Рабочий ритм таков: перед запуском второй сессии – коммит; после каждого принятого результата – коммит. Пусть их будет много и мелких: история изменений не музей, её не надо украшать.

**После параллельной работы – сверка, а не вера.** Обе сессии закончили, обе отчитались, всё выглядит хорошо. Теперь единственный достоверный источник – состояние файлов, и смотреть надо его: что изменилось суммарно, живы ли обе правки, прогоняются ли тесты. И правило перепроверки через время: к важной правке вернуться через полчаса и убедиться, что она всё ещё на ме-

сте. Звучит параноидально ровно до первого случая, когда окажется, что уже нет.

## Отраслевой ответ: у каждой задачи свой каталог

Для случаев, когда параллельность нужна всерьёз и надолго, у отрасли есть решение получше очереди, и о нём стоит знать даже до того, как оно понадобится.

Идея простая: развести сессии по копиям проекта, вместо раздела по задачам. Система контроля версий умеет создавать несколько рабочих каталогов одного проекта – каждый со своей копией файлов, но с общей историей. Сессия работает в своём каталоге и физически не может затереть чужой: файлы разные. Когда задача готова, её результат вливается в общую историю осознанным действием – вы смотрите, что вливается, и конфликт, если он есть, разрешается на ваших глазах, а не молча в момент сохранения.

Переводя на язык картинок: вместо двух поваров у одной разделочной доски – две доски и общая кладовая, куда готовое сдают через окошко. У окошка стоите вы.

Для вечернего проекта это обычно избыточно – правила из предыдущего раздела покрывают его с запасом. Но знать, что приём существует и как называется («рабочие деревья», worktree, в терминах git), полезно: в тот день, когда вы захотите гонять три задачи параллельно, вы будете искать решение по имени, а не изобретать.

## Агент в фоне

Отдельный случай нескольких рук, который выглядит иначе, а устроен так же: долгая задача, запущенная в фоне. Вы поручили агенту большую работу – прогнать проверку по всем страницам, переделать сотню записей – и занялись другим. Рук снова две: ваша и фоновая, и фоновая финиширует в непредсказуемый момент.

Два правила на этот случай. Первое: фоновая задача получает свою территорию так же, как параллельная сессия, – границей в постановке. Второе, менее очевидное: у фоновой работы должен быть однозначный признак завершения и место, куда она кладёт результат, – файл отчёта, запись, отметка. Иначе получается фигура, которой в этой книге отведена отдельная глава: работа, о состоянии которой ничего не известно. «Кажется, оно ещё работает» и «кажется, оно закончило» – два состояния, в которых нельзя принимать решения.

И примета времени, чтобы глава не выглядела страшилкой: параллельность – это хорошо. Агенты снимают главное ограничение одиночки – последовательность: впервые один человек может вести три работы одновременно, как маленькая команда. Вся глава сводится к тому, что вместе с возможностями команды приходят и обязанности бригадира: делить территорию, вести общую историю, сверять результат. Обязанности небольшие. Возможности – нет.

> **Если вы заказчик.** Если над вашим проектом работает несколько исполнителей – людей, агентов, вперемешку, – один вопрос вскрывает зрелость процесса: «как вы делите работу, чтобы не затирать друг друга, и как вливаете результаты?» В ответе должны прозвучать территории и осознанное слияние. Ответ «мы аккуратно» вы теперь умеете переводить сами.

> **Упражнение.** Устройте конфликт нарочно, на учебном проекте, – лучше один раз увидеть. Откройте два окна агента. В первом попросите изменить заголовок страницы списка, во втором, не дожидаясь, – изменить в том же файле текст под заголовком. Примите оба результата и откройте файл: с высокой вероятностью одна из правок исчезла. Посмотрите историю (`git log`), найдите потерянное в разнице между версиями (`git diff`) и верните файлы к последней зафиксированной точке командой `git checkout -- .`. Весь опыт занимает десять минут

> и оставляет то, что не оставляет никакое чтение: память о том, как выглядит молчаливая потеря – и как легко она чинится, когда есть история.

Осталась одна глава – о том, что всем этим рукам поручать. Она короткая, потому что рабочего правила там на страницу: делайте то, что сможете проверить сегодня вечером.

## Рамка: что поручать всем этим рукам

Про то, каким должен быть объём первой версии, написаны целые книги, и у этой темы есть неприятное свойство: чем больше о ней читаешь, тем меньше делаешь. Поэтому здесь будет короткая глава с одним рабочим правилом, уже прозвучавшим в конце прошлой: **делайте то, что сможете проверить сегодня вечером.**

Правило выглядит слишком простым для решения, о котором столько спорят. Разберём, почему оно работает – и почему именно с агентами оно стало важнее, чем было когда-либо.

### Почему рамку рвёт именно сейчас

У прежнего разработчика-одиночки был встроенный ограничитель аппетита: скорость рук. Задумать можно было что угодно, но каждый пункт задуманного стоил вечеров, и список хотелок редел сам собой, об усталость.

Агент этот ограничитель снял. Попросить «ещё и личный кабинет, и уведомления, и статистику» стало так же дёшево, как не попросить, – и код появится, много кода, быстро. Ограничитель переехал в другое место, и вся эта книга последовательно показывала куда: в проверку. Сгенерировать можно сколько угодно; проверить – сколько успеете. Всё, что сгенерировано и не проверено, вы уже видели в первый вечер: это картинка, у неё нет обязательств.

Отсюда и правило вечера. Объём шага определяется не фантазией и не скоростью генерации – ёмкостью вашей проверки. Что сможете сегодня прогнать руками и тестом, то сегодня и существует. Остальное – план.

## Три строки вместо технического задания

Рамка первой версии – или второй, или любого следующего рывка – умещается в три строки, и все три вам уже знакомы по отдельности:

**Кто один.** Один пользователь, чьей рукой вы будете проверять. Для сервиса обращений это был «человек, у которого что-то случилось». Попытка угодить двоим сразу – заявителю и менеджеру, покупателю и складу – удваивает даже не работу: количество непроверенного.

**Что одно.** Один сценарий от начала до конца: отправил – сохранилось – увидели – отметили. Сквозной путь – к нему мы ещё вернёмся, когда речь пойдёт о проверках; здесь он служит мерой объёма: версия – это один проходимый путь; набором возможностей она станет позже, по одному за раз.

**Чем проверите.** Критерий готовности, сформулированный до начала работы, проверяемый и скучный: «обращение с сайта доходит до списка и переживает перезапуск». Вы уже знаете из главы о постановке, что этот пункт – проектирование краёв; здесь добавим второе его свойство: он останавливает. Версия, у которой критерий выполнен, готова – и всё, что чешется доделать, идёт в следующую.

Три строки плюс список «что не входит» из части 0, который к этому месту книги дорос до постоянного инструмента: письменная граница снимает зуд, потому что отложенное решением перестаёт быть отложенным по слабости.

## Ловушка «ещё одной функции»

Главный враг рамки не лень и не нехватка сил – прилив сил. Всё работает, агент послушен, вечер молод, и рука сама тянется: раз так легко, добавим ещё фильтры. И роли. И красивое оформление. И вход по паролю.

Механику вы уже знаете по частям книги, соберём её в одно место. Каждая добавка приносит свои края, которые надо продумать; свои проверки, которые надо написать и гонять; своих соседей, которых она задевает; свои состояния, которым нужно место в отчёте. Функции складываются, а их стоимость перемножается – и наступает вечер, когда проект перестаёт помещаться в вашу проверку целиком. С этого вечера вы больше не знаете, работает ли он. Знаете только, что работал.

Проекты новичков умирают от добавления многократно чаще, чем от нехватки. Недоделанный проект с одним работающим сценарием живёт: им пользуются, его развивают. Переделанный, где семь сценариев по половине, доживает до первого настоящего пользователя.

## Как выглядит рамка, которой не было

Обещанный пример из нашей практики – с необычной стороны. В последней части будет разговор об отпущенном направлении: группе сайтов, которую мы свернули, потому что месяцы работы не привели ни одного человека. Забежим вперёд и посмотрим на ту историю глазами этой главы: что было бы, поставь мы рамку в начале.

Рамка звучала бы так: один сайт, один сценарий – человек ищет услугу, находит страницу, оставляет заявку, – критерий: столько-то настоящих переходов и хотя бы одна заявка за месяц. Проверка стоила бы один сайт и один месяц. Вместо этого строилась сеть: ге-

нерация статей, карты сайтов, перелинковка – работа на месяцы, добротная, местами изящная. Проверка предположения «люди придут этим путём» откладывалась, потому что строить было интереснее, чем проверять, – знакомый мотив? Это «ещё одна функция», только на уровне целого направления. Когда проверка наконец случилась, она стоила всего построенного.

Годится эта история как масштабная линейка: ловушка добавления одинаково устроена на кнопке, на функции и на группе сайтов. Меняется только цена вечера, в который выясняется правда.

> **Если вы заказчик.** Предложение подрядчика на двадцать пунктов – повод для одного вопроса: «какой из пунктов проверяет главное предположение – что этим вообще будут пользоваться – и можно ли начать с него одного?» Хороший исполнитель ответит, какой и как. Уклончивый ответ означает, что проверка предположения отложена в конец, – а вы теперь знаете, сколько она стоит в конце.

> **Упражнение.** Опишите вторую версию своего вечернего сервиса тремя строками: кто один, что одно, чем проверите. Всё остальное, что просилось в неё, – в список «не входит», письменно. Потом сверьте объём с правилом вечера: успеете проверить сегодня? Если нет – режьте сценарий, пока не влезет. Занимает десять минут.

На этом ремесло собрано целиком. Вы умеете ставить задачу и держать её в границах, читать результат и не верить отчётам, копить правила в файле, распоряжаться вниманием и силой моделей, дирижировать несколькими руками и – с этой главы – выбирать, что вообще делать. Инструмент освоен.

Дальше – часть II, инженерия, и у перехода простая логика: ремесло позволяет получать нужный результат, инженерия – его не терять. Система контроля версий как страховка от всего, что вы про-

чли про параллельные руки. Проверки, которые охраняют проект, пока вы спите. Данные, секреты и выкладка на настоящий сервер. И последнее звено – чтобы всё это давало результат: чтобы вас нашли, вам поверили и вы это видели. Всё то, из-за отсутствия чего вечерние проекты остаются вечерними.

## Часть II. Инженерия

Ремесло из первой части позволяет получать от агента нужный результат. Инженерия, которой посвящена вторая, отвечает за то, чтобы полученное не терялось: не пропало, не сломалось молча, не открыло дверь посторонним.

Скажем прямо, что вас ждёт: набор предохранителей. История изменений, проверки, устройство данных, секреты, воспроизводимая среда, выкладка. Каждый предохранитель стоит около часа сегодня и экономит день потом; ни один не делает проект лучше на вид. Все вместе они решают, доживёт ли проект до второго месяца – по нашим наблюдениям, вечерние проекты умирают именно там, и умирают по-инженерному: от потерянного, сломавшегося молча и открытого настежь.

И одно число, чтобы задать вес всей части. Большие исследования год за годом показывают: код, сгенерированный моделями, проходит проверки безопасности примерно в половине случаев, и эта доля не растёт со сменой поколений – при том что синтаксическая корректность давно выше девяноста пяти процентов. Генерация решена. Всё остальное – предмет этой части, и отвечает за него владелец.

## История изменений всерьёз

Git сопровождает вас с первого вечера, и до сих пор мы обходились четырьмя командами: завести историю, добавить, зафиксировать, посмотреть состояние. В части про несколько рук добавились ещё три – посмотреть журнал, посмотреть разницу, вернуть файлы к последней точке. Этого достаточно, чтобы не терять работу. Пришло время закрыть тему всерьёз. Дело даже не в новых командах – их будет одна. Дело в правильном понимании, зачем всё это.

Расхожее представление о системе контроля версий – «архив для командной работы»: штука, которая нужна, когда людей много, а пока ты один, это бюрократия. Представление устойчивое и перевёрнутое. Один плюс агент – это уже не один, вы знаете это с главы о нескольких руках. А архив – самая скучная из функций истории. Настоящих функций три, и все три нужны лично вам, команда тут ни при чём.

**История – это машина времени.** Любое состояние проекта, которое вы зафиксировали, достижимо обратно. Отсюда следствие, которое меняет характер работы сильнее любого приёма из этой книги: эксперимент перестаёт быть риском. Можно позволить агенту большую переделку, зная, что путь назад – одна команда. То, что у опытных людей выглядит смелостью, обычно устроено проще: у них есть выход.

**История – это показания.** Когда что-то сломалось, первый вопрос – «что менялось перед этим?». Без истории на него отвечает память, и отвечает плохо. С историей – журнал: вот изменения, вот даты, вот подписи. Разница между «кажется, я вчера что-то трогал в форме» и «в 19:40 изменены два файла, вот построчная разница» – это разница между гаданием и расследованием.

**История – это граница между сделанным и болтающимся.** Зафиксированное защищено: от агента, от второй сессии, от вас

завтрашнего. Незафиксированное не защищено ни от чего. Мы повторяли это в разных формах всю первую часть; здесь эта мысль становится определением: в проекте существует то, что в истории. Остальное – черновик на ветру.

## Наш антипример, без прикрас

Признание, которое обязано быть в этой главе, потому что без него она звучала бы поучением с горы.

У нас в хозяйстве годами жила другая культура – резервные копии файлов. Перед правкой файл копируется рядом с суффиксом: имя, точка, «резерв», дата. В одной нашей служебной папке таких копий десятки – слои правок за месяцы, как годовые кольца. А одна из наших рабочих служб не под системой контроля версий по сей день: у неё вместо истории – эти самые копии, и в её файле правил стоит специальная строка о том, как с этим жить.

Чем это плохо, мы выяснили на себе, по пунктам. Копия отвечает на вопрос «что было» – и молчит на вопросы «что именно изменилось», «когда» и «зачем». Сравнивать копии между собой – ручная работа. Понять, которая из шести копий та самая, – археология. А когда над файлами работает несколько сессий – а вы знаете по главе о нескольких руках, как это бывает, – копии превращаются в единственный носитель затёртого труда, и вся защита держится на случайности: сделал кто-то копию перед правкой или нет.

История изменений закрывает всё это одним механизмом. Мы это знаем, мы этим живём – и всё равно тащим хвост старой привычки, потому что перевод работающей службы под контроль версий требует остановки и аккуратности, а «пока и так работает». Узнаёте интонацию? Это «потом разберусь» из первой главы книги. У нас тоже есть свои.

Мораль без морализаторства: копии с суффиксами – это история изменений, собранная вручную из худших материалов. Если вы делаете их – вы уже признали, что история нужна. Осталось взять нормальный инструмент.

## Четыре действия, которых хватит на год

Весь рабочий набор помещается в четыре действия. Три вы знаете, четвёртое новое.

**Фиксировать после каждого работающего шага.** Правило вечера из части 0, теперь с уточнением про подпись. Подпись коммита – сообщение себе в будущее, и качество этого сообщения проверяется одним вопросом: поможет ли оно через месяц понять, что здесь произошло? «Правки» и «фикс» не помогут. «Поиск: пустой запрос показывает весь список» – поможет. Пишите подпись как строку из журнала, который будете читать при расследовании, – потому что ровно им она и станет.

**Смотреть разницу перед фиксацией.** `git diff` перед `git add` – двадцать секунд, за которые вы видите, что на самом деле уходит в историю. Это последний рубеж, на котором ловятся сюрпризы от агента: лишний файл, правка за границей задачи, случайно затянутый секрет. Про последнее будет отдельная глава, и привычка смотреть разницу – половина её содержания.

**Возвращаться.** Три случая, по нарастающей серьёзности. Испортили незафиксированное – `git checkout -- .` возвращает файлы к последней точке; вы делали это в упражнении про конфликт двух окон. Нужно откатить уже зафиксированное – `git revert` создаёт новый коммит, отменяющий выбранный: история при этом остаётся целой: отмена дописывается сверху, и это правильный путь. Нужно посмотреть, как выглядел проект неделю назад, – история позволяет перенестись в любую точку и вернуться. Механику двух

последних не обязательно помнить наизусть: достаточно помнить, что они существуют, – команду подскажет агент, а вот о существовании выхода в панике не вспоминают, если не знали о нём заранее.

**Ветка под опасную работу.** Новое действие, и единственное по-настоящему новое понятие главы. Ветка – это параллельная линия истории: вы отходите от рабочего состояния в сторону, делаете там что угодно – большую переделку, рискованный эксперимент, – и рабочая линия остаётся нетронутой. Получилось – влили в основную. Провалилось – выбросили ветку целиком, основная и не заметила. Завести: `git switch -c peredelka`. Вернуться на основную: `git switch main` – или `master`, если ваша основная называется так; какая у вас, покажет `git branch`. Всё.

Ветка – это машина времени, направленная вперёд: вместо возврата после аварии вы заранее делаете так, что аварии негде случиться. Правило, когда она обязательна, простое: если словами задачи является «переделать» или «попробовать» – работа идёт в ветке. Достройка по маленькому шагу живёт и на основной линии.

## Что это меняет в работе с агентом

Соберём главу в одно наблюдение. Все приёмы первой части – границы, маленькие шаги, проверки – работали на то, чтобы агент не наделал лишнего. История изменений добавляет второй контур: даже если наделал, это обратимо. Два контура вместе дают то состояние, в котором с агентом можно работать смело: страховка спереди, страховка сзади.

Инженерия, как мы сказали в начале части, – набор предохранителей. Этот – первый, самый дешёвый и самый недооценённый. Ни одна из следующих глав без него не работает: проверки бессмысленно чинить без возможности откатиться, секреты нельзя вычи-

стить без понимания истории, выкладка без фиксированных точек – прыжок без страховки.

> **Если вы заказчик.** Попросите показать историю изменений проекта за последний месяц – не код, просто журнал: даты, подписи, объёмы. Настоящая история при активной работе выглядит как десятки записей с внятными подписями. Пустая или редкая история при бурной деятельности означает, что работа идёт мимо неё – руками на сервере, копиями файлов, – и всё, что вы прочли в этой главе про возврат и показания, к вашему проекту не относится.

> **Упражнение.** Заведите ветку и устройте в ней погром. `git switch -c pogrom`, затем попросите агента «переделать страницу списка радикально, на свой вкус» – ровно та задача, которую вся первая часть учила не давать. Посмотрите на результат, прогоните тесты, ужаснитесь или удивитесь. Потом вернитесь: `git switch main` – и убедитесь, что ваш проект стоит нетронутым, как будто ничего не было. Это упражнение учит двум вещам сразу: что ветка делает эксперименты бесплатными и что «радикально на свой вкус» – не задача. Обе стоят того, чтобы прожить их на себе.

Дальше в части – проверки: как один тест из части 0 вырастает в набор, который охраняет проект, пока вы занимаетесь другим. История и проверки вместе образуют ту пару, на которой стоит вся остальная инженерия: одна помнит, другие сторожат.

## Проверки, которые работают за вас

У вашего проекта с первого вечера есть один тест: обращение отправлено – обращение сохранилось. Он уже отработал своё жалованье не раз: ловил учебную поломку, охранял границу правки,

краснел в упражнении с погромом. Пришло время превратить одиночку в службу охраны – и по дороге разобраться, что вообще стоит охранять.

Сначала – зачем это, двумя фразами. Ручная проверка не масштабируется: к десятому сценарию полный обход занимает полчаса, которые вы перестанете тратить примерно на третий день. Автоматической не лень – она отработает после каждой правки, ночью, в чужих руках, и не стоит вам ни минуты внимания, пока зелёная.

## С какого конца строить набор

В учебниках по тестированию рисуют пирамиду: много мелких проверок отдельных функций внизу, немного крупных сквозных наверху. Для команды с большим проектом это разумно. Для вечернего проекта мы советуем противоположное, и совет этот из практики: начинайте с самого широкого теста и дробите только там, где он перестаёт справляться.

Самый широкий – это сквозной путь главного сценария: тот самый, ради которого сервис существует. Для обращений: отправить через форму – найти в списке. Один такой тест проверяет разом форму, приём, сохранение, чтение и показ – пять слоёв одним прогоном. Мелкий тест отдельной функции проверяет функцию; сквозной проверяет, что сервис делает свою работу. Когда придётся выбирать, что написать первым, – выбор всегда за сквозным.

Наша собственная система живёт ровно на этом принципе, и мы назовём числа, чтобы был виден масштаб. Основной вид проверки у нас – обход настоящих страниц с проверкой содержимого: на один недавно построенный раздел приходится тридцать пять проверок, на соседний сервис – девять, полный прогон занимает минуты. Модульных тестов, любимых учебниками, у нас единицы. Это осознанное решение для нашего масштаба, и у него есть граница, назовём её прямо: когда внутри появляется сложная логика с ветв-

лениями – расчёты, права, состояния, – ей нужны свои мелкие тесты, потому что сквозной прогон не доберётся до каждой ветки. У вечернего проекта такой логики обычно ещё нет.

Из наших же прогонов – правило, которое важнее любой пирамиды: **проверка, которая гоняется дольше, чем вы готовы ждать, не гоняется вообще.** Набор, работающий три минуты, запускают после каждой правки. Набор, работающий полчаса, запускают перед сном, потом по пятницам, потом никогда. Скорость набора – условие его существования, удобство тут дело десятое.

## Тест как контракт

Теперь главная мысль главы – та, из-за которой разговор о тестах идёт в книге про работу с агентами, хотя место ему, казалось бы, в учебнике по инструментам.

В первой части вы ставили агенту границы словами: «имена не переименовывай», «сохранение не трогай». Слова работают, пока агент их помнит, – а помнит он, как вы знаете, в пределах одной сессии, и то не всегда. Тест делает границу физической. Имя поля, поведение при пустом вводе, само существование сохранения – всё, что зафиксировано тестом, агент не может изменить тихо: тест покраснеет, и вы узнаете об этом через минуту, а через месяц, из жалобы человека, уже не узнаете, потому что узнали сейчас.

Это меняет статус тестов в проекте. Тест – письменный договор о том, что в проекте считается правдой; «проверка качества» тут слишком слабое слово. Каждый тест – пункт договора: обращения сохраняются; пустая форма не проходит; обработанное уходит вниз списка. Агент, нарушивший пункт, ловится машиной, без вашего участия. Помните формулу из главы о постановке – «названия часть договора, и договор пишет владелец»? Тест – это тот же договор, только с исполнением: нарушение не обсуждается – оно загорается.

Отсюда практическое правило для работы с агентом, замыкающее круг из части I: просите агента прогонять тесты после каждой правки – эта строка уже стоит в вашем файле правил – и не принимайте результат с красным. Красный после правки означает ровно одно из двух: либо агент вышел за границу (чаще всего), либо ваш договор устарел и его надо менять осознанно. Оба случая требуют вашего решения, и оба всплыли вовремя.

## Три теста, которые окупаются первыми

Расширим ваш набор с одного до трёх. Больше на этом этапе не нужно; эти три закрывают то, что ломается первым.

**Сквозной путь** – уже есть с части 0: отправить, найти в списке.

**Пустой и кривой ввод.** Пятый вопрос первого вечера – «что увидит человек, если поле пустое» – закрывается ровно этим тестом. Что происходит, когда форму отправили без имени или вовсе пустой? Это первый край, о который спотыкаются настоящие люди, – и первое, что агент забывает, если не спросить. Тест короткий:

```
def test_pustaya_forma_ne_prohodit():
    spisok_do = client.get("/spisok").text

    otvet = client.post("/otpravit", data={})
    assert otvet.status_code != 200

    spisok_posle = client.get("/spisok").text
    assert spisok_do == spisok_posle
```

Смысл: пустая отправка не притворяется успешной, а список до и после неё совпадает – мусорная запись не появилась. Обратите внимание на приём со снимком «до»: тест не делает предположений о том, что уже лежит в базе, он сравнивает состояние с самим собой. Наша первая версия этого теста предполагала пустую базу – и падала на исправном сервисе, потому что сосед-тест успевал со-

здать запись раньше. Тесты не должны зависеть от порядка, в котором их гоняют, и снимок «до» – самый простой способ этого добиться.

И заметьте, чего тест не требует: никакого конкретного красивого поведения. Только минимум – не соврать и не намусорить. Красивое сообщение об ошибке закажете агенту потом, отдельной задачей с границей.

**Состояние.** Отметка «обработано» делает своё дело: тест создаёт обращение, отмечает, проверяет, что статус изменился. Это охрана самой хрупкой части сервиса – той, где данные меняются.

Три теста, секунды на прогон, и у проекта появилось то, что можно всерьёз называть охраной: путь, края, изменение состояния. Всё остальное достраивается по мере роста – по правилу «сломалось то, что не было покрыто, – покрой и живи дальше».

## Два ритуала

Первый вы знаете с первого вечера, теперь он получает статус закона: **новый тест обязан покраснеть у вас на глазах.** Сломайте то, что он проверяет, увидьте красное, верните. Тест, который ни разу не краснел, не проверен сам – а к чему приводят непроверенные проверки, мы подробно разберём в части про эксплуатацию, на собственной истории, которая нам дорого обошлась.

Второй ритуал новый, и он про дисциплину в неприятный момент. Рано или поздно тест покраснеет «не вовремя»: вы заняты другим, правка вроде хорошая, а этот красный мешает. Соблазн – отключить его «на время». Так вот: у отключённого «на время» теста нет ни одного известного случая обратного включения. Выбор в этот момент настоящий, и он бинарный: либо разобраться сейчас – красный тест показывает либо поломку, либо устаревший пункт договора, и то и другое важно, – либо удалить тест осознанно, комми-

том с подписью, почему договор расторгнут. Отключение – это удаление, которое врёт, что оно временное. А набор, в котором есть «ну этот красный, это нормально», перестаёт быть охраной целиком: следующий настоящий красный утонет в привычном.

> **Если вы заказчик.** Вопрос «есть ли у вас тесты» бесполезен: ответ всегда «да». Работают два других. «Сколько времени занимает полный прогон и когда он запускался последний раз?» – рабочий набор гоняется ежедневно и укладывается в минуты. И «покажите, как он краснеет»: пусть при вас сломают что-нибудь и прогонят. Зелёный набор, который не умеет краснеть, вам в этой книге ещё встретится.

> **Упражнение.** Доведите набор вечернего проекта до трёх тестов из этой главы – сквозной у вас есть, два оставшихся поставьте агенту как задачи по всем правилам части I, с границей и критерием. Потом главное: устройте поломку, которая уронит ровно два теста из трёх, – например, сломайте сохранение. Посмотрите на выдачу: два красных, один зелёный, и по тому, какие именно, видно, где поломка. Это второй навык чтения тестов – по рисунку красного ставить диагноз, не открывая код.

История помнит, проверки сторожат. Следующая глава – про то, что они охраняют на самом деле: данные. Их особенность в том, что они переживают любой код, накапливают любые ошибки и не прощают решений, принятых мимоходом, – включая те, которые за вас принял столбец со значением по умолчанию.

## Данные переживут код

В главе о чтении результатов мы назвали слой данных самым дорогим в исправлении и обещали объяснение. Вот оно, целой главой, и

открывается оно неравенством, на котором держатся все решения этой главы.

Код можно переписать. Целиком, с нуля, хоть трижды – старый выбрасывается, новый занимает его место, и через минуту никто не вспомнит, каким был прежний. Данные переписать нельзя. Их можно только переносить – аккуратно, с сохранением каждого пришедшего обращения, потому что обращения написали люди, и заново их не напишет никто. Код – ваша работа, её можно повторить. Данные – след чужих действий, они невоспроизводимы.

Отсюда правило, переворачивающее интуицию новичка: решения о данных – самые важные решения проекта. Как называется поле, что в нём хранится, что стоит по умолчанию – всё это переживёт любой код, который вокруг напишут и перепишут. Агент предложит структуру за секунды; проверять её стоит внимательнее всего остального, потому что цена переделки здесь измеряется переносом накопленного, а не перегенерацией.

## Структура и её изменение

Структура хранения – по-взрослому «схема» – это решение о том, из чего состоит запись: у обращения есть имя, контакт, текст, время, статус. Вы приняли это решение в первый вечер, в задаче агенту, и с тех пор оно молча держит на себе всё: форму, список, тесты.

Пока проект живёт, структура догоняет жизнь: понадобится телефон отдельным полем, пометка «откуда пришёл», приоритет. И здесь появляется единственное новое понятие главы – миграция. Слово звучит серьёзнее, чем предмет: миграция – это изменение структуры, записанное отдельным действием, которое можно выполнить, проверить и повторить на другой копии базы. Противоположность миграции – «я там руками поправил в базе»: изменение, которое нигде не записано, ни на какой другой копии не воспроизведётся и через месяц станет загадкой.

Для вечернего проекта миграция – это маленький файл:

```
# migracia_001_telefon.py
import sqlite3

db = sqlite3.connect("zayavki.db")
db.execute("ALTER TABLE obrasheniya ADD COLUMN telefon TEXT
NOT NULL DEFAULT ''")
db.commit()
print("Поле telefon добавлено")
```

Запустили один раз – поле появилось, старые записи получили пустое значение, ничего не потерялось. Именно один раз: при повторном запуске миграция откажет с ошибкой «такой столбец уже есть», и это нормальное поведение, даже полезное – она отказывается применяться дважды. Не пугайтесь, встретив этот отказ: он означает, что дело уже сделано. Файл остаётся в проекте и в истории изменений: это документ о том, что структура менялась, когда и как. Задача агенту на такое изменение ставится по всем правилам части I, с одним обязательным добавлением в критерий проверки: «существующие записи сохраняются и получают такое-то значение нового поля». Про судьбу старых записей агент сам не спросит – а это, как сейчас увидите, главный вопрос.

## Значение по умолчанию принимает решения за всех

История, ради которой написана эта глава. Числа настоящие.

В одном из наших проектов у профилей людей есть настройка «показывать мои контакты на публичных страницах». Хорошая, правильная настройка – каждый решает сам. В июле при проверке выяснилось: настройка включена у восемнадцати тысяч четырёхсот семидесяти четырёх профилей из восемнадцати тысяч четырёхсот семидесяти пяти. У одиннадцати с половиной тысяч из них был заполнен телефон, и он отдавался на публичных страницах любому желающему.

Восемнадцать тысяч человек не принимали такого решения. Его не принимал и никто из нас: не было ни совещания, ни строчки в задаче, ни момента, когда кто-то сказал «показываем контакты всех». Решение приняла одна строка в описании структуры – значение по умолчанию у столбца, поставленное при его создании. Каждый новый профиль получал «показывать: да» просто потому, что так был устроен столбец, и восемнадцать тысяч молчаливых «да» накопились сами, без единого злого умысла.

Исправляли аккуратно, в обратную сторону: согласие сброшено всем, телефоны в данных сохранены – люди смогут включить показ сами, теперь уже действительно сами. Значение по умолчанию перевёрнуто, формы перестали его подставлять.

Правило из этой истории стоит запомнить дословно: **значение по умолчанию – это решение, которое вы принимаете за всех будущих пользователей сразу.** Каждый, кто не выберет сам, получит ваше «по умолчанию» – а не выбирают почти все. Поэтому у каждого умолчания в структуре должен быть ответ на вопрос «почему именно так», и для всего, что касается приватности, согласий и денег, ответ по умолчанию один: выключено, пока человек не включил. Обратное умолчание в этих областях – юридический риск, который вы создаёте одной строкой и накапливаете молча, со скоростью регистраций.

И заметьте, как это связано с началом главы. Ошибку в коде мы бы поправили правкой. Ошибка в умолчании столбца за месяцы превратилась в восемнадцать тысяч записей, каждая из которых выглядит как осознанный выбор человека, – и отличить настоящие «да» от автоматических уже невозможно. Данные накапливают последствия решений, включая те, которых никто не принимал.

## Две проверки, о которых забывают

**Перезапуск.** Ритуал из первого вечера, повторим одной строкой, потому что он относится сюда: данные, которые не пережили остановку и запуск сервиса, данными не являются. Проверяется за минуту, входит в привычку навсегда.

**Одновременность.** Вопрос, который не приходит в голову, пока сервисом пользуется один человек: что произойдёт, если двое отправят форму в одну и ту же секунду? Сохранятся оба? Не перепутаются ли номера? Для вашего сервиса в нынешнем виде ответ благополучный: база выстраивает записи в очередь сама. Но вопрос обязан войти в ваш список, потому что ответ не всегда благополучный, а проверяется он просто – спросите агента: «что произойдёт при двух одновременных отправках, покажи, где это обрабатывается». Знакомый приём маршрута из части I, применённый к самому хрупкому месту. Сервисы падают на одновременности ровно тогда, когда начинают быть нужными, – в момент наплыва.

Остался третий вопрос о данных – где лежит их копия и переживут ли они смерть диска. Ему посвящена отдельная глава в части про эксплуатацию.

> **Если вы заказчик.** Один вопрос из этой главы стоит задать про любой сервис, который собирает данные людей: «какие настройки приватности и согласий стоят по умолчанию – и почему?» Ответ «по умолчанию всё включено, так удобнее» означает, что подрядчик создал вам юридический риск и назвал его удобством. Правильное умолчание для согласий – выключено; всё остальное должно иметь письменное обоснование. Проверить можно без техники: зарегистрируйтесь на собственном сервисе и посмотрите, на что вы «согласились», не нажав ничего.

> **Упражнение.** Добавьте в свой сервис поле «телефон» через миграцию из этой главы – но сначала создайте пару обращений, чтобы было чему выживать. Прогоните миграцию, откройте список: старые обращения на месте, у них пустой телефон, новые сохраняются с телефоном. Потом второй заход, важнее первого: попросите агента добавить поле «согласие на ответ по почте» – и посмотрите, какое значение по умолчанию он выберет, если вы не скажете. Наш опыт подсказывает, что ответ вас заинтересует. Исправьте на правильное – теперь вы знаете, какое оно.

Данные в порядке: структура записана, умолчания осознаны, изменения оформлены миграциями. Следующая глава – про то, что охраняет вход в эти данные и в весь проект: секреты и права. Там будут три наши истории по нарастающей и одно правило, которое дороже всех замков: права проверяются попыткой, а не чтением настроек.

## Секреты и права

Глава о безопасности в книге для начинающих обычно выглядит как парад страшилок: взломы, хакеры, чёрные ходы. Наша будет скучнее и полезнее, потому что нам есть на что опереться: мы проверяли собственное хозяйство и нашли там настоящие дыры, без всякого учебника. Ни одна из них не была хитрой уязвимостью. Все три были одного сорта: вещь, которую никто не проверил, потому что «ну там же наверняка нормально».

Самая частая дыра проектов, собранных с агентом, устроена именно так. Это ключ, лежащий в открытом виде, и права, которых никто не проверял. Обе темы закрываются за вечер, и обе – про привычку, а не про знания.

## Секреты: одно правило и пять строк

Секрет – это любая строка, знание которой даёт доступ: пароль базы, ключ платёжного сервиса, токен бота, которым ваш сервис пишет вам в мессенджер. Правило обращения с ними одно, и оно абсолютное: **секрет живёт в отдельном файле настроек и никогда не попадает в историю изменений.**

Почему так строго – из-за свойства истории, которое в главе про git мы хвалили: она помнит всё. Секрет, хотя бы раз зафиксированный в коммите, остаётся в истории навсегда – даже если следующим коммитом вы его стёрли. Любой, у кого окажется копия репозитория, достанет его из прошлого одной командой. Отсюда и второе правило, неприятное: секрет, побывавший в истории, считается утёкшим. Лечится это только заменой самого секрета – выпустить новый ключ, отозвать старый. Стереть след нельзя, можно лишь сделать след бесполезным.

Механика защиты – пять строк. Файл `.env` в корне проекта, в нём настоящие значения:

```
TG_TOKEN=здесь-настоящий-токен
```

Строка в `.gitignore`, чтобы файл никогда не попал в историю:

```
.env
```

И файл-образец `.env.example` – та же структура, пустые значения, живёт в истории как документация:

```
TG_TOKEN=
```

Код читает значения из окружения – вы уже делали это с именем файла базы в части 0, приём тот же. Одна тонкость: сам по себе файл `.env` процесс не читает, его содержимое надо загружать в

окружение при запуске. Это делается готовой маленькой библиотекой или парой строк – поставьте агенту задачу «загружай .env при старте» с проверкой «значение из файла видно сервису». Новый человек – или вы на новой машине – копирует образец в `.env`, вписывает свои значения, и всё работает. Осталась одна привычка, замыкающая защиту: `git diff` перед фиксацией, знакомый вам по главе об истории. Агент может вписать ключ прямо в код – «чтобы работало»: злодейства тут нет, есть кратчайший путь. Ваша задача – поймать это глазами до коммита, и просмотра разницы для этого достаточно.

Насколько это распространённая беда, видно по отраслевым числам. Репозитории, где включён кодовый помощник, содержат утёкшие секреты заметно чаще обычных – примерно на сорок процентов. За прошлый год на публичных площадках нашли около двадцати девяти миллионов свежих секретов, лежащих прямо в коде, и утечки ключей к сервисам ИИ выросли на восемьдесят один процент – новые ключи потекли раньше, чем сложились привычки их хранить. Агент прекрасно пишет код. Хранить тайны он не помогает.

## Три наших случая, по нарастающей

Теперь обещанные дыры из собственного хозяйства. Мы расскажем их лестницей, потому что каждая следующая была больнее предыдущей, а мораль у всех одна.

**Ключи в коде.** При разборе одного служебного скрипта обнаружилось, что в нём прямо в тексте лежат токен бота, ключ к внешнему сервису и пароль базы данных. Скрипт старый, написан в спешке, «потом переделаем». Дальше вы знаете правило: просто вынести ключи в настройки уже недостаточно – они считаются утёкшими, и всё найденное отправилось на замену. Час работы, которого не было бы, займи вынос ключей пять минут в день написания.

**Замок, который не запирал.** Настраивая резервное копирование, мы положили служебный архив в облачное хранилище с закрытыми правами на самом файле – и, по правилу из этой главы, тут же проверили попыткой. Проверка показала: архив скачивался без всякого пароля. Права на файле были закрыты. Но у хранилища была своя политика доступа, уровнем выше, и она перекрывала права файла. Мы смотрели на один замок и не знали о существовании второй двери. Дыру закрыли в ту же минуту, хранилище для копий завели отдельное и закрытое – но урок остался: без привычки проверять попыткой мы бы узнали о второй двери не от себя.

**Хранилище, открытое на запись.** Самый тяжёлый случай, и нашли мы его по следам предыдущего, проверяя уже всё подряд. Политика одного из хранилищ разрешала анонимному пользователю запись и удаление. Кто угодно из интернета мог загрузить туда что угодно – или стереть всё: больше восьми гигабайт работ, загруженных людьми. Проверено практикой: пробная анонимная запись прошла успешно, пробное анонимное удаление тоже. Эта настройка досталась нам вместе с проектом и существовала неизвестно сколько – до того дня никто не проверял её попыткой.

Мораль одна на три случая, и мы держим её как закон: **права проверяются попыткой, а не чтением настроек.** Настройки говорят, как должно быть. Попытка показывает, как есть. «Там закрыто» – это гипотеза, как отчёт агента из части I; документом является отказ, полученный при настоящей попытке войти без ключа, записать без права, скачать без пароля. Такая попытка занимает минуту и не требует никакой квалификации – только привычки её делать.

## Права внутри приложения

Вторая половина темы – права внутри самого сервиса: кто что может. И здесь надо знать особенность агентов, подтверждённую

большими исследованиями: сгенерированный код проходит проверки безопасности примерно в половине случаев, и эта доля не растёт от поколения к поколению моделей. Самая типичная пропажа – ровно проверка прав: код исправно делает то, что просили, для всякого, кто попросит. Страница «отметить обращение обработанным» из вашего проекта выполнит отметку для любого, кто откроет адрес, – потому что в задаче не было слов «а кто, собственно, имеет право отмечать».

Пока сервис учебный и смотрит только на вас, это допустимо – и это допущение стоит записать в список ограничений, он для того и существует. Но правило должно поселиться в голове до того, как появятся настоящие пользователи: **у каждого действия, которое меняет данные, должен быть ответ на вопрос «кто имеет право это делать» – и проверка этого ответа в коде.** Агент сам не спросит. Спросить – ваша строка в задаче, и в файле правил ей самое место: «для действий, меняющих данные, всегда спрашивай меня, кто имеет право».

> **Пять вопросов о безопасности,** на которые владелец сервиса отвечает без подготовки. Где лежат секреты и нет ли их в истории изменений? Кто может писать в хранилище – проверено ли попыткой? Какие действия меняют данные и кто имеет на них право? Что записывается в журнал – увидите ли вы чужую попытку? Что вы будете делать, если ключ утечёт, – и сколько времени займёт замена? Пятый вопрос особый: ответ на него готовят заранее, потому что искать его в момент утечки поздно.

> **Если вы заказчик.** Из всей главы вам достаточно двух вопросов. «Проверялись ли права доступа попыткой – покажите, как выглядела проверка». И «что произойдёт, если ключ от такого-то сервиса утечёт, – сколько времени займёт замена». Первый отличает настроенную безопасность от предполагае-

мой. Второй показывает, готовились ли к плохому дню заранее. Слова «у нас всё закрыто» без показанной попытки вы уже умеете переводить.

**Упражнение.** Два действия по этой главе. Первое: заведите в проекте `.env` и `.env.example` по образцу, перенесите туда имя файла базы, добавьте `.env` в `.gitignore` – и проверьте попыткой: `git status` не должен показывать `.env` как новый файл, а `git check-ignore .env` должен подтвердить, что файл под запретом. Второе, интереснее: попросите агента «добавь отправку уведомления в мессенджер о новом обращении» – и посмотрите, куда он положит токен. Если в код – вы только что поймали самую массовую дыру индустрии у себя дома, бесплатно и до последствий. Перенесите в `.env` и добавьте строку в файл правил.

Секреты убраны, права проверены попыткой. Следующая глава – о месте, где всё это работает: чем «работает у меня» отличается от «работает у людей», что такое контейнер и почему мы после двух лет на визуальном конструкторе переписали всю автоматизацию на обычный код – история, в которой агент сместил баланс сильнее, чем нам казалось.

## Где живёт запущенный проект

Есть фраза, которую произносил каждый, кто что-нибудь собирал: «у меня работает». Обычно с интонацией законного недоумения – работает же, сами смотрите. Эта глава о том, почему фраза ничего не гарантирует и как сделать, чтобы гарантировала.

«У меня работает» – это свойство вашей машины, а не проекта. На вашей машине стоит нужная версия языка, установлены пакеты, лежит файл настроек, задана переменная окружения, о которой вы

забыли через час после того, как задали. Проект работает не сам по себе – он работает в среде, и среда эта собиралась месяцами, незаметно, слоями. Перенесите проект на другую машину – и выяснится, сколько слоёв не переехало; мы это проверили дорого. Между «работает у меня» и «работает где угодно» лежит отдельная инженерная задача, и имя ей – воспроизводимость.

## Воспроизводимость: среда как список

Задача решается по-разному на разном масштабе, и мы пойдём от простого.

Минимальный уровень, обязательный для всех, у вас уже почти есть: среда описана словами. README перечисляет команды установки и запуска; `.env.example` перечисляет настройки; список пакетов лежит в проекте. Проверка этого уровня вам знакома по упражнению «разверни с нуля»: если по описанию проект поднимается на чистой машине – среда воспроизводима, все слои названы.

Уровень выше – среда описана исполняемо: контейнер. Слово звучит технологичнее, чем предмет. Контейнер – это упакованная среда: в одном месте записано, какая нужна система, какие пакеты, какие настройки, и из этой записи собирается одинаковая коробка на любой машине. Разница с README принципиальная: README читает и выполняет человек, ошибаясь и пропуская, а запись контейнера выполняет машина, одинаково всегда. «У меня работает» превращается в «работает в коробке, а коробка собирается где угодно».

Вечернему проекту контейнер не обязателен, и мы не будем изображать обратное: на одном компьютере с одним пользователем README и `.env.example` закрывают вопрос. Контейнер становится нужен, когда проект переезжает на сервер – там он наводит порядок, которого иначе не добиться. Но одно правило о контейнерах

нужно знать до всякого сервера, потому что оно про сохранность, и мы за него заплатили.

У контейнера есть внутренность – и она одноразовая. Коробку пересоздают: при обновлении, при перезагрузке, при переезде – и всё, что жило только внутри неё, исчезает вместе со старой коробкой. Поэтому у всего, что должно уцелеть, должно быть постоянное место снаружи: данные, загруженные людьми файлы, настройки. Внутри коробки живёт только сам проект, который собирается заново из записи. Правило звучит очевидно ровно до того дня, когда выясняется, что правки целого дня существуют только внутри работающей коробки, – с нами такое случилось, и полный рассказ об этом впереди, в части про эксплуатацию. Пока запомните вопрос, который надо задавать любому контейнеру: что в тебе умрёт вместе с тобой?

## Почему мы ушли от конструктора

Теперь обещанная история про выбор, где вообще жить автоматизации. Она наша собственная, заняла три месяца и стоит того, чтобы рассказать её с числами.

Два года наша автоматизация жила в визуальном конструкторе – инструменте, где рабочий процесс собирается мышью из кубиков: «получить данные», «преобразовать», «отправить». Выбор был правильный: конструктор дал скорость тогда, когда не было ни агентов, ни навыка писать код. Процессов накопилось около шестидесяти – публикации, сборы данных, уведомления, целая фабрика в кубиках.

К началу лета из шестидесяти работающих осталось два. Остальные умерли, переехали или превратились в загадки. А в июле мы вывели конструктор совсем: два последних процесса переписаны обычными скриптами, порты закрыты, кубики – в архив. Причины сто-

ит перечислить по журналам, как было, потому что каждая – урок про целый класс инструментов, конкретное имя тут ни при чём.

Песочница не выпускала наружу. Внутри конструктора код исполняется в ограниченной среде: в нашей версии из неё не было доступа к переменным окружения – тем самым, где по прошлой главе живут секреты. Ключи приходилось протаскивать окольными путями, и каждый окольный путь – это дыра из главы о секретах, встроенная в архитектуру.

Кубики не лежат в истории изменений. Процесс правится мышью, и после правки нет ни разницы «было – стало», ни подписи, ни возможности вернуться. Вся первая глава этой части – машина времени, показания, граница сделанного – к конструктору неприменима: у него нет истории, только текущее состояние и память человека, который клацал.

Проверок нет и прогона вхолостую нет. Проверить процесс можно одним способом – запустить по-настоящему и посмотреть. Что это означает для дисциплины, вы понимаете по главе о тестах: контракта нет, границу держать нечем, каждый запуск – эксперимент на работающем.

Сторожа не переживали перемен. Наша канарейка – сигнал, следящий, что процесс жив, – после одного из переездов выла триста пятьдесят восемь раз подряд по процессу, который давно умер и был заменён. Триста пятьдесят восемь сообщений о покойнике – это уже не наблюдение; чем кончаются тревоги, на которые невозможно реагировать, разберёт часть про эксплуатацию.

И главное, из-за чего чаша окончательно склонилась: пришли агенты. Агент пишет код на обычном языке блестяще – и совершенно беспомощен в чужом визуальном конструкторе: он не может ни прочитать процесс из кубиков, ни надёжно его поправить, ни прогнать тесты, которых нет. Конструктор, который годами был

ускорителем, за один сезон превратился в тормоз: всё, чему учит эта книга – задачи с границами, тесты как контракт, история, файл правил, – работает с кодом и не работает с кубиками.

Что дал переезд, одной строкой на пункт: каждый бывший процесс стал скриптом – под историей изменений, с тестом, с прогоном вхолостую, с журналом, который можно читать. Те самые предохранители этой части, все разом, просто сменой места жительства.

Урок для вас в этой истории неожиданно оптимистичный. Нам конструктор был нужен, потому что в своё время он был самым быстрым способом начать. У вас этого этапа нет: агент даёт ту же скорость старта сразу в коде – со всеми предохранителями, которые к коду прилагаются. Ваш вечерний сервис тому доказательство: он собран за вечер и при этом под историей, с тестами и настройками по правилам. Два года назад такое сочетание скорости и порядка было недоступно никому.

> **Если вы заказчик.** Вопрос «на чём это построено» стоит задавать ради одного свойства, названия тут вторичны: «покажите, как проект разворачивается на чистой машине, и где лежит история его изменений». Решения на конструкторах и платформах-сборщиках проверяйте тем же вопросом – у многих из них ответ «никак и нигде», и это значит, что всё, что вы прочли в этой части про историю, тесты и воспроизводимость, к вашему заказу не относится. Иногда это осознанный компромисс ради скорости. Осознанным он становится, только если вы задали вопрос.

> **Упражнение.** Составьте опись среды своего проекта: один файл, где перечислено всё, что нужно для его жизни, – версия языка, команды установки, содержимое `.env.example`, команда запуска, команда тестов. Потом проверьте опись делом – в новой папке, по списку, ничего не вспоминая. Всё, что пришлось

> вспомнить сверх написанного, – дыра в описи, допишите. Этот файл – будущий фундамент и для контейнера, если он понадобится, и для комплекта передачи: среда, существующая списком, переносима; среда, существующая привычкой, умирает вместе с машиной.

Осталось вывести проект к людям – и следующая глава про этот шаг: чем выкладка отличается от «загрузить файлы», какие решения она требует и на какие грабли здесь наступили мы, включая правку, которая доехала до сервера и не доехала до посетителей.

## Выкладка

Всё это время ваш сервис жил на адресе, начинающемся со ста двадцати семи, – домашнем адресе, который видите только вы. Выкладка – день, когда у проекта появляется настоящий адрес и настоящие посетители. В голове новичка это «загрузить файлы на сервер». На деле это серия решений, и у каждого есть цена. Хорошая новость: решений конечное число, и все они собираются в чеклист на одну страницу, которым глава и закончится.

### Что меняется в момент выкладки

Меняется всё, что было допущением. Дома сервис слушал только вас – теперь его открывает кто угодно, включая роботов, которые начнут стучаться в первые же часы, не спрашивая разрешения. Дома ошибка показывала подробности на экране – наружу подробности показывать нельзя: текст ошибки рассказывает про устройство проекта больше, чем вы хотели бы рассказать постороннему. Дома соединение было простым – наружу оно обязано быть защищённым: замочек в адресной строке давно перестал быть украшением, без него браузеры пугают посетителей, а формы предупреждают о небезопасности.

И первая ловушка, о которой надо сказать прямо, потому что в неё попадает каждый второй: сервер, которым вы пользовались всю книгу, – отладочный. Команда с флагом перезагрузки, которую вы запускали каждый вечер, создана для разработки: она удобная, разговорчивая и совершенно не предназначена держать настоящих посетителей – про это прямо написано в её собственной документации, которую никто не читает. Для выхода наружу тот же проект запускается рабочим сервером – это другая команда, а не другой код, и агент настроит её по одной просьбе: «подготовь запуск для настоящих посетителей, отладочный режим выключи». Проверить, что просьба выполнена, просто: сломайте что-нибудь и посмотрите на страницу ошибки – посетительская версия молчит о подробностях.

## Выкладка как повторяемое действие

Главный принцип главы, и он вам уже знаком по духу всей части: выкладка – это скрипт. Ритуал из ручных шагов – временное состояние, которое надо пережить один раз.

Первая выкладка руками простительна. Вторая руками – уже риск: шагов десяток, забыть один – нормальная человеческая ошибка, и проявится она не сразу. У нас выкладка одного из проектов – это один скрипт, который делает всё сам: доставляет файлы, проверяет, что код цел, обходит десяток настоящих страниц и убеждается, что они отвечают, фиксирует состояние в истории. Ручные шаги оттуда изгонялись по одному, и у каждого изгнания была причина – однажды пропущенный шаг. Скрипт не забывает, не устаёт и делает одиннадцатую выкладку так же, как первую; человек – нет.

Для вечернего проекта скрипт выкладки пишет агент – это задача по всем правилам части I, и в критерий проверки просится главное: «после выкладки скрипт сам открывает главную страницу и прове-

ряет, что сервис отвечает». Выкладка без проверки после – это отправленное письмо без вопроса «дошло?».

Теперь три наших грабли, чтобы вам не собирать свои.

**Файл без версии в адресе.** Браузеры запоминают загруженные файлы, чтобы не качать их заново, – и это прекрасно ровно до первой правки: вы выложили новую версию, а часть посетителей ещё дни получает старую из собственного кеша. Мы наступили на это всерьёз: правка доехала до сервера и не доехала до людей. Лекарство – пометка версии в адресе подключаемого файла: изменилась пометка – браузер берёт свежее. Агенту это ставится одной строкой в файл правил.

**Выкладка целиком поверх чужого.** Если на сервере живёт что-то, чего нет в вашей папке, – чужой раздел, загруженные людьми файлы, соседний проект, – выкладка «всё целиком» это чужое сотрёт или перекроет. У нас однажды в дереве сборки лежал посторонний файл, и полная выкладка сломала бы весь сайт, – спасло то, что выкладывали точечно. Правило: скрипт выкладки явно знает, что он кладёт и чего не трогает, и список «не трогать» в нём записан, как граница в задаче агенту.

**Пометка «не трогаем».** Файл, исключённый из выкладки с пометкой «не трогаем, он особенный», – это мина замедленного действия: он живёт без истории, без пути доставки исправлений, и однажды переживает всех, кто помнил, почему он особенный. Правило здесь: исключение из выкладки – это долг, и ему место в списке ограничений с датой, а не в чьей-то памяти.

## Закон приходит вместе с посетителями

Раздел, которого нет в других книгах этого жанра, а должен бы быть в каждой. Пока сервис жил на домашнем адресе, он был вашим личным делом. С настоящими посетителями он становится

делом публичным – и на него начинает действовать законодательство. Мы не юристы и заключений не даём; наша задача скромнее – назвать вопросы, для которых «мы про это не подумали» перестаёт быть ответом ровно в момент выкладки.

**Где стоит сервер.** Если ваш сервис собирает данные россиян – а форма обращений с именем и контактом уже собирает, – закон требует хранить эти данные на серверах в России. Строгость тут вовсе не теоретическая: один из наших проектов, с данными восемнадцати с половиной тысяч человек, жил на зарубежном сервере – так исторически сложилось, и это было прямым нарушением. Переезд на российский сервер стал отдельной операцией со своими потерями, о которых расскажет часть про эксплуатацию. Урок дешевле нашего: страна сервера выбирается до того, как в базу упадёт первая настоящая запись. После – это уже переезд, а не выбор.

**Политика и согласие.** Форма, собирающая хоть что-то о человеке, требует двух вещей: документа на сайте, объясняющего, что вы делаете с данными, и согласия человека на это. Про согласие вы уже знаете главное из главы о данных – его умолчание должно быть выключенным. Добавим второе требование, из главы о правах: согласие должно реально сохраняться, и проверяется это попыткой – отправьте форму с отметкой и найдите отметку в базе. У нас есть история о согласии, которое выглядело собранным и не сохранилось ни разу, – она впереди. Шаблон политики вам подготовит агент, а вот проверить, что согласие доезжает до хранилища, можете только вы.

**Домен.** Требования к владельцам доменов растут – подтверждение личности через государственные сервисы уже становится нормой для российских зон. Практическое следствие одно, и вы встретите его снова в разговоре о передаче проектов: домен оформляется на владельца проекта, на его настоящие данные. Домен «на под-

рядчика» или «на кого получилось» – это мина, которая срабатывает при первом же ужесточении правил или первой ссоре.

Раздел получился короткий, и это осознанно: подробности меняются, вопросы остаются. Держите три вопроса – где сервер, есть ли политика с настоящим согласием, на кого домен – в чек-листе каждой выкладки, и большую часть неприятностей этого рода вы не встретите никогда.

### Закрыт – значит закрыт осознанно

Последний сюжет главы – про промежуток, когда проект уже снаружи, но ещё не готов к гостям.

Это штатное состояние: сайт собирается, наполняется, проверяется на настоящем адресе – а поисковикам и случайным прохожим там пока делать нечего. Для этого существует вежливая табличка «закрыто»: файл, который просит роботов не заходить, и пометка на страницах «в указатели не вносить». У нас такой режим – нормальная часть запуска, и один из проектов прямо сейчас стоит закрытым, дожидаясь наполнения.

Опасность у таблички одна: о ней забывают. Сайт готов, посетителей ждут, реклама запущена – а табличка «закрыто» висит, и поисковики послушно обходят сайт стороной месяцами. Обнаруживается это обычно вопросом «почему нас никто не находит», и хорошо, если быстро. Правило: закрытость – это запись в списке ограничений с условием снятия («открыть, когда будет двадцать страниц»), и у неё, как у всякого немого состояния, должен быть сторож – хотя бы строка в чек-листе выкладки: «проверить, не закрыт ли сайт, если он должен быть открыт».

> **Чек-лист выкладки.** Семь пунктов на страницу, вешается рядом с рабочим местом. 1. Рабочий сервер вместо отладочного, ошибки наружу молчат. 2. Соединение защищено. 3. Секре-

ты в настройках сервера, в истории их нет. 4. Скрипт выкладки: кладёт своё, не трогает чужое, проверяет после. 5. Версии в адресах подключаемых файлов. 6. Закон: сервер в нужной стране, политика на месте, согласие сохраняется, домен на владельце. 7. Табличка «закрыто» снята – или висит осознанно, с условием снятия.

**Если вы заказчик.** Два вопроса при приёмке выкладки. «Покажите, как выглядит выкладка новой версии» – ответ должен быть скриптом или хотя бы записанной последовательностью, а слова «ну, копируем файлы» означают, что каждая выкладка – лотерея. И «в какой стране стоит сервер и на кого оформлен домен» – два факта, которые дешевле всего узнать до подписания и дороже всего после расставания.

**Упражнение.** Выложите вечерний проект туда, где его увидит другой человек, – подойдёт самый простой сервер за пару сотен рублей в месяц; агент проведёт вас по шагам, а чек-лист главы удержит от пропусков. Потом попросите кого-нибудь пройти главный сценарий с телефона и посмотрите на два места: дошло ли обращение до вашего списка – и что человек сказал про скорость и удобство. Первый настоящий посетитель на настоящем адресе даёт больше знаний о проекте, чем неделя домашней отладки. Обращение от него, кстати, останется в вашей базе первым настоящим – берегите его.

Осталась последняя глава части – о том, ради чего всё это делалось: чтобы сервис давал результат. Работающий и приносящий пользу – разные состояния, и между ними лежит система из трёх звеньев: вас должны найти, вам должны поверить, и вы должны видеть, что происходит.

## Чтобы это дало результат

Часть II начиналась с обещания, что инженерия решает, доживёт ли проект до второго месяца. Все главы до этой держали обещание в одном смысле: проект не потеряется, не сломается молча, не откроет дверь посторонним. Последняя глава – про второй смысл, ради которого всё затевалось: проект должен приносить пользу, ради которой его делали. Заявки, читателей, покупателей, сэкономленные часы – у каждого своё, но у всех это называется одним словом: результат.

Работающий сервис и сервис, дающий результат, – разные состояния. Между ними лежит система из трёх звеньев: вас должны найти, вам должны поверить, и вы должны видеть, что происходит. Глава пройдёт по звеньям в общих чертах – принципами, которые переживут смену алгоритмов и сервисов, – а в конце покажет, как эти же звенья выглядят на взрослом масштабе, на нашей фабрике.

### Звено первое: вас должны найти

Прекрасный сервис по адресу, которого никто не знает, эквивалентен отсутствию сервиса. Находят вас двумя путями, и с недавних пор их действительно два, а не один.

Первый – классический поиск. Его механика для владельца маленького сайта сводится к трём условиям. Страница отвечает на конкретный вопрос конкретными словами – теми, которыми спрашивают люди, а не теми, которыми красиво говорить о себе: «полив газона осенью» находится, «комплексные решения для вашего участка» не находится никогда. Описание страницы – та пара строк, что видна в выдаче, – написано для человека, который решает, куда нажать. И карта сайта существует и не врёт: это файл-указатель, по которому поисковик узнаёт, что у вас есть. Указатель умеет врать, мы проверили: однажды наша карта из-за ошибки в

сборке схлопнулась до одной ссылки на целый сайт – и месяцами сообщала поисковикам, что кроме главной страницы у нас ничего нет, при исправном сайте и зелёных отчётах сборки. Заметили при ручной сверке, чинили пересбором. Урок в духе всей книги: у карты, как у всякой проверки, должна быть своя проверка – число ссылок в ней против числа настоящих страниц.

Второй путь новый: ИИ-помощники. Всё больше людей вместо поиска спрашивают – и получают готовый ответ со ссылками на источники. Попасть в эти источники – отдельная задача со своими правилами, и хорошая новость в том, что правила эти прямее классических: помощники цитируют страницы, где ответ дан прямо, фактами, в первых абзацах, без предисловий «в современном мире». Жанр называется у нас «цитируемый блок»: короткий фактический ответ на вопрос в начале страницы, дальше подробности. Плюс маленькая новинка, которую почти никто пока не сделал: файл-визитка для ИИ-помощников, где сайт сам рассказывает, о чём он и каким фактам можно верить, – стоит полчаса, ставится агентом, у большинства соседей отсутствует.

Общий принцип звена: находимость – свойство текста, никакой магии продвижения в ней нет. Страница, по делу отвечающая на настоящий вопрос, со временем находится; страница про «инновационные решения» не находится ни при каком продвижении.

### Звено второе: вам должны поверить

Нашли – читают. И здесь решает качество текста, про которое важно сказать главное: это инженерная величина, а не вкусовщина. Мы пришли к этому, поставив тексты на конвейер, и правила оттуда переносятся на любой масштаб, включая одну страницу вашего сервиса.

Первое правило вы узнаете: канон стиля – это файл правил для текстов. Та же механика, что в главе про файл правил проекта: оса-

док редакторских ожогов, записанный так, чтобы исполнитель – человек или модель – подчинялся без напоминаний. Какие слова запрещены, как звучат заголовки, что с тире и точками, чего в текстах не бывает никогда. Наш канон вырос до внушительного документа, и почти за каждой строкой – место, где текст звучал фальшиво, и мы разобрались почему.

Второе правило дороже: факты проверяются гейтом. Модель, генерящая текст, – тот же агент из главы о чтении результатов: она достраивает недосказанное правдоподобно, и с цифрами, датами и нормативами делает это особенно уверенно. У нас между генерацией и публикацией стоит проверка: отдельная модель-судья читает текст и ищет утверждения, которых нет в источниках; спорное уходит человеку, с выдуманными цифрами текст не публикуется. Правило для вашего масштаба то же, что было с кодом: **сгенерированный текст – гипотеза; факты в нём проверяются до публикации, и цифры берутся только из своих данных.** Текст с одной настоящей цифрой бьёт текст с пятью округлыми – и по доверию читателя, и, всё чаще, по цитируемости у ИИ-помощников, которые учатся отличать фактуру от воды.

### Звено третье: вы должны видеть

Здесь короче всего, потому что механику вы уже знаете из этой книги – осталось направить её на результат.

У контентной системы тот же главный вопрос, что у сервиса: какое событие должно происходить регулярно и кто заметит, если оно перестанет? Показы, переходы, заявки – это события, и на их отсутствие ставится тот же сторож, что на обращения. А самый дешёвый прибор – два числа рядом: сколько пришло и сколько сделало нужное действие. Врозь эти числа успокаивают; рядом – ставят диагноз. Много показов при нуле переходов – не о том пишем или не

так называемся; переходы при нуле заявок – читают и не верят, смотрите звено второе.

И правило решений, за которое мы заплатили месяцами работы: **по числам, а не по ощущениям.** В главе о рамке мы забегали вперёд к направлению, которое строилось увлечённо и не привело ни одного человека; как принималось решение о его судьбе, доскажет последняя часть. Правило, ради которого мы его сюда позвали, короткое: ощущение «дело идёт» не отчитывается о результате. Числа отчитываются.

## Как это выглядит на взрослом масштабе

Обещанная витрина – наша фабрика, в общих чертах, без единого названия сервиса. Показываем её ради одного: она собрана из тех же деталей, которые вы уже держали в руках.

Ежедневно фабрика делает примерно следующее. Собирает темы – из запросов, которые люди задают поиску, и из наблюдения за отраслью; запас тем сторожится датчиком входа. Пишет тексты – модели по канону стиля. Проверяет – судья ищет выдуманные факты, спорное уходит человеку; это гейт, без зелёного света которого публикации нет. Публикует по расписанию – с ручным подтверждением там, где выход публичный: правило «подтверждено плюс прошедшее время означает немедленную отправку» стоит в нашем файле правил жирным, и вы догадываетесь, что оно там не с рождения. Замеряет – позиции и показы приезжают по расписанию в базу, и датчики смотрят на них так же, как сторож на обращения: пропали данные – сигнал, просели показы – сигнал. Возвращает – то, что просело или выстрелило, влияет на завтрашние темы.

Тот же конвейер в другом материале собирает видео: сценарий, кадры, озвучка, сборка, публикация – с теми же гейтами, включая проверку кадров глазами до оживления, потому что у видео свои способы наврать. И всё это – скрипты под историей изменений, с

тестами и прогонами вхолостую: часть про уход от конструктора вы читали две главы назад, фабрика и есть то, что переехало.

Файл правил, гейт с проверкой фактов, расписание с подтверждением, сторожа входа и выхода, история изменений – ни одной детали, которой не было в этой книге. Масштаб другой – детали те же. Это и есть главная мысль витрины: система, дающая результат, не требует секретных знаний. Она требует, чтобы обычные детали были собраны в замкнутый круг: найти – поверить – увидеть – поправить.

Потому что цельность здесь решает всё. Страница без ответа на вопрос не находится. Найденная без фактов не убеждает. Убедительная без замера не улучшается – вы даже не узнаете, что она убедительна. Три звена, и слабейшее определяет результат целого; у системы из двух звеньев результат обычно нулевой, зато отчётность красивая.

> **Если вы заказчик.** Вся глава сжимается в два вопроса подрядчику, который делает вам «сайт с продвижением»: «по каким запросам нас находят – покажите» и «сколько из нашедших оставляет заявку – покажите». Два числа, показанные рядом, отличают систему от набора страниц. Если в ответ звучит «мы наращиваем присутствие» без чисел – вы покупаете второе звено без первого и третьего, а чему равна такая система, сказано абзацем выше.

> **Упражнение.** Напишите для своего сервиса одну страницу-ответ – например, «как оставить обращение и что с ним будет дальше»: это настоящий вопрос вашего будущего пользователя, заданный его словами. Ответьте в первом абзаце прямо и фактами, дальше – подробности. Опубликуйте на своём настоящем адресе. Потом проверьте звено в деле: задайте свой вопрос ИИ-помощнику и посмотрите, чем он ответит и кого про-

> цитирует. Сегодня, скорее всего, ещё не вас – а через месяц спросите снова. Этот замер, повторяемый раз в месяц, и есть третье звено в миниатюре.

И последнее, прежде чем закрыть часть. Откройте шесть вопросов из первого вечера – те, что вы записали после формы, говорившей «Спасибо». К этому месту книги у вас есть ответ делом на каждый: где данные – в базе за миграциями; что после обновления и перезапуска – проверено тестом; что при ошибке – есть тест кривого ввода; что уходит на сервер – видно в истории; что увидит человек – решено вами, вплоть до умолчаний; запускается ли по инструкции – опись среды проверена в чистой папке. Шесть вопросов новичка выросли в инженерию, и дальше они продолжат расти.

На этом инженерия собрана. Ваш проект под историей, под охраной тестов, данные структурированы, секреты убраны, среда воспроизводима, выкладка повторяема, и у него есть система, ведущая к результату. По всем меркам этой книги он перестал быть черновиком – он стал настоящим.

И именно поэтому дальше начинается самое интересное. Настоящий проект живёт: им пользуются, его меняют, он стареет и ломается – как правило, тихо. Часть III – про эксплуатацию: про проверки, которые врут, тишину, которую никто не слушает, правки, которые сносят соседей, и вопрос, где на самом деле лежит ваш проект. Это самая дорогая часть книги – в пересчёте на то, что мы заплатили за её материал.

# Часть III. Эксплуатация

## Проверка, которая врёт

В первый вечер этой книги вы собрали форму, которая говорила «Спасибо» и выбрасывала обращение. Тогда это выглядело детской ошибкой из мира новичков: ну кто в здравом уме выпустит такое наружу. Вся вторая часть учила вас проверять – руками, тестами, перезапуском – именно затем, чтобы подобное не выживало.

Пришло время рассказать, как та же самая поломка несколько месяцев прожила у нас, с двумя десятками сайтов и написанными проверками. Мы разберём этот случай подробно, по дням, с настоящими цифрами – по одной причине: чужие такие истории никто не показывает, а учиться на приглаженных примерах бесполезно.

Тема этой части книги – эксплуатация: всё, что происходит с проектом после того, как он заработал. И начинается она с самого коварного отказа из всех возможных. Ломается сама проверка. У сломанной проверки нет симптомов: она зелёная.

### Диагноз: нет спроса

Двадцать первое июля. Повод для разбора звучал буднично: в сводке по заявкам с сайтов подозрительно пусто, надо пройти весь путь заявки и понять, что происходит.

Путь заявки (у нас на сайтах это заявка от возможного заказчика – то же, что обращение в вашем вечернем сервисе, только путь длиннее) – цепочка, и дальше мы будем называть её трактом, как называли в своих записях. Человек заполняет форму на сайте, запрос уходит на сервер приёма, запись ложится в базу, уведомление улетает в мессенджер, ночная синхронизация переносит запись в об-

щую базу, оттуда её читает сводка. Разбор прошёл по этой цепочке звено за звеном и нашёл настоящие поломки.

Самая крупная: мост между двумя базами стоял мёртвым с апреля. Заявки исправно ложились в первую базу, а сводка читала вторую, куда с весны ничего не приезжало. Поэтому на экране были нули при настоящих записях в хранилище. Мост починили, нули сменились числами. Заодно нашлись дыры поменьше: на одной из форм не было защиты от мусорных отправок, а пара задач по расписанию была запущена без блокировок и могла накапливать зависшие процессы.

К вечеру разбор завершился выводом. Тракт исправен: тестовая заявка проходит от формы до уведомления, все звенья отвечают, сводка показывает правду. Числа по месяцам: май – 14 заявок, июнь – 4, июль – 0. В мае шла рекламная кампания, в июне она закончилась. Заключение напрашивалось само: система в порядке, обнулился входящий поток. Нет спроса.

Обратите внимание, как крепко стоит этот вывод. Он сделан после разбора, в котором нашлись и починились настоящие поломки – значит, смотрели внимательно. Он подтверждён сквозной тестовой заявкой. Он объясняется внешней причиной с датами: была кампания – были заявки, кончилась – кончились. Это вывод, с которым не стыдно идти к владельцу.

И он был неверным.

## Через четыре дня

Двадцать пятое июля. Другая сессия, другой вопрос, заданный владельцем почти мимоходом: а формы вообще на всех сайтах стоят как надо? Вопрос был не про тракт, тракт четыре дня как проверен. Вопрос был про то, что стоит перед трактом.

Первая находка. На шести доменах сети формы на главной странице не существовало. Вместо неё стояла кнопка «Оставить заявку», которая вела к нужному месту страницы – а этого места на странице не было. Нажатие прокручивало страницу в никуда. Человек, пришедший с рекламы или из поиска, видел нормальный сайт с нормальной кнопкой, нажимал – и ничего. Сколько людей развернулось на этом месте, мы не узнаем никогда: несуществующая форма не оставляет следов.

Вторая находка. За отправку форм на восьми доменах отвечал общий сценарий – небольшая программа, которая работает в браузере посетителя, перехватывает нажатие кнопки и показывает вежливое «спасибо» вместо перезагрузки страницы. Этот сценарий искал формы по служебному признаку – а этого признака не было ни в одном шаблоне ни одного сайта. То есть сценарий был подключён, загружался, работал – и за всё время не обработал ни одной настоящей формы, потому что ни одна под его условие не подходила. Формы, которые всё-таки существовали, отправлялись напрямую, без него, и человек после нажатия кнопки видел вместо «спасибо» строку служебного ответа сервера. Заявка при этом, к слову, доходила – но выглядело это как поломка, и часть людей наверняка считала именно так.

Третья находка, самая тихая из всех. Тот же сценарий отправлял согласие на обработку персональных данных под одним именем поля. Сервер приёма читал другое имя. Стороны никогда не совпали. Согласие не сохранялось вообще никогда – даже у тех людей, которые нашли форму, поставили флажок и отправили заявку. Снаружи всё выглядело образцово: флажок есть, без него форма не отправляется, в базе есть столбец для согласия. В столбце всегда лежал ноль.

И четвёртая, из-за которой три первые смогли прожить так долго. Сам этот сценарий не лежал в системе контроля версий. Он жил

прямо на рабочем сервере, куда его когда-то положили руками, а в инструменте выкладки напротив него стояла пометка «не трогаем». Файл, который отвечает за все формы сети, не имел ни истории изменений, ни хозяина, ни пути, по которому до него доехало бы исправление. Вдобавок он подключался на страницах без версии в адресе – даже после починки часть посетителей ещё долго получала бы из кеша браузера старый вариант.

## Что это значило

Сложим обе картины. Разбор двадцать первого июля был добросовестным и нашёл настоящие поломки. Но он проверял тракт – цепочку от точки приёма заявки и дальше. А путь человека начинается раньше: открыть главную, найти форму, заполнить, нажать, поверить. На шести сайтах этот путь обрывался на втором шаге. Тракт был исправен ровно в том смысле, что всё, что в него попадало, доходило до конца. Просто на шести доменах в него нечем было попасть.

Диагноз «нет спроса» был поставлен по исправному тракту и убедительным числам. Прими мы его окончательно – решение ушло бы в маркетинг: поднять бюджет, поменять объявления, искать новые каналы. Деньги потекли бы в рекламу, которая приводит людей на страницу с кнопкой в никуда. Чем убедительнее ошибочный диагноз, тем дороже он обходится – именно потому, что по нему начинают действовать.

Стоит сказать и о том, чего в этой истории нет. В ней нет злодея и нет грубой небрежности. Каждая из четырёх поломок по отдельности объяснима, и вместе они складываются в обычную энтропию работающего проекта: шаблоны меняются, имена полей расходятся, служебные пометки переживают всех, кто помнил, зачем они поставлены. Эксплуатация – это дисциплина, которая исходит из

того, что такая энтропия есть всегда, и строит сети, в которые она попадается.

## Почему проверка была зелёной

Разберём механику. Четыре причины, по которым внимательный разбор с настоящими находками поставил неверный диагноз.

**Проверяли систему, а путь человека начинается раньше системы.** Разбор шёл от точки приёма: вот сюда прилетает запрос, дальше посмотрим каждое звено. Всё, что до этой точки – страница, кнопка, форма, сценарий отправки, – в проверку не попало, потому что не ощущалось частью системы. Это ощущение обманчиво. Для человека с той стороны экрана системой является всё, включая кнопку, которая никуда не ведёт. Сквозная проверка начинается с первого щелчка настоящего посетителя, с чистого браузера, без служебных закладок – иначе она сквозной только называется.

**Проверочная площадка показывала зелёное по построению.** У сети есть площадка предпросмотра, где страницы смотрят перед выкладкой. Адреса служебных запросов там ведут на другую службу, и та отвечает отказом – это нормально и известно. Но форма была написана так, что показывала «Спасибо» независимо от ответа. В итоге на предпросмотре формы «работали» всегда: заполнил, нажал, увидел благодарность. Зелёный результат этой проверки не значил ничего – она была не способна покраснеть ни при каких обстоятельствах. Отсутствие проверки хотя бы никого не успокаивает; такая проверка успокаивала всех.

**Выводы делались по тексту исходников, а не по поведению.** Тот же день, двадцать пятое июля, преподал и обратный урок – мы дважды объявили поломкой то, что работало. На одной странице висела служебная пометка, выглядевшая как след незавершённой работы, и по ней страницу записали в сломанные – а она принимала заявки исправно, пометка была мёртвым мусором. На

другом сайте поиск не нашёл флажка согласия под ожидаемым именем – а флажок был, просто поле называлось иначе: другая система, другие имена. Оба приговора вынесены по чтению текста. Правило симметрично, и его стоит запомнить в обе стороны: по тексту исходников нельзя заключить ни что сломано, ни что работает. Заключение даёт только запуск и наблюдаемое поведение.

**Тишину никто не слушал.** Ноль заявок в июле не разбудил никого, потому что все сигналы тревоги были настроены на события: ошибка – сообщение, отказ – сообщение. Ноль событий ошибкой нигде не считается, это отсутствие, а на отсутствие сигналов обычно не ставят. Между тем именно отсутствие ожидаемого – главный симптом тихих отказов. У нас этот урок к тому моменту уже накопился в нескольких экземплярах: копии данных не делались трое суток – молча; служба напоминаний пять суток копила зависшие процессы – молча; очередь материалов на проверку старела почти месяц – молча. Тихая деградация оказалась основным видом отказа вообще, притом что мы считали её редкостью. Сигнал «ожидаемое не происходит уже N дней» ловит целый класс поломок, невидимых для сигналов об ошибках – включая нашу кнопку в никуда.

Тишина оказалась темой настолько отдельной, что ей отведена вся следующая глава. Пока достаточно запомнить, что ноль событий – это тоже событие.

> **Если вы заказчик.** Два вопроса подрядчику стоят больше, чем весь остальной разговор о приёмке. Первый: «Как вы проверяете, что заявка с сайта доходит до менеджера?» Хороший ответ описывает сквозной путь и заканчивается словами «вот запись в базе». Второй: «Если заявки перестанут доходить – кто и через сколько узнает?» Ответ «мы проверяли при сдаче» переводится так: узнаете вы, от несостоявшегося клиента, через месяц. Оба вопроса не требуют от вас технических знаний – только готовности дослушать ответ до конца.

> **Упражнение.** Прямо сейчас, не откладывая: откройте свой сайт как посетитель, в чистом окне браузера, и отправьте через форму настоящую заявку. Дойдите до конца пути – найдите её в базе, в почте, в мессенджере, там, куда она должна приходить. Увидеть «Спасибо» недостаточно: вы теперь знаете цену этому слову. Если весь путь прошёлся и заявка нашлась – поздравляем, вы потратили три минуты. Если нет – вы только что сэкономили себе июль, который мы разбирали всю эту главу.

Форма, с которой начиналась эта книга, врала одному человеку – вам, и прожила до первого обновления страницы. Проверка, о которой шла речь здесь, врала системе, которая ей доверяла, и прожила месяцы. Устроены обе поломки одинаково. Разница в сроке жизни, и объясняется она просто: первой было негде спрятаться, вторую охраняло зелёное. В следующей главе мы посмотрим, как устроить наблюдение за сервисом, при котором зелёному можно верить: что мерить, где ставить сигналы и почему датчик на выходе конвейера ничего не говорит о его входе.

## Кто заметит тишину

Прошлая глава закончилась неудобным выводом: зелёная проверка может врать, и врёт она тише всех остальных поломок. Возникает естественный вопрос – а что тогда вообще считать признаком здоровья? Если «ошибок нет» ничего не гарантирует, на что смотреть?

Ответ у этой главы короткий, и он поначалу кажется странным: признак здоровья – присутствие ожидаемых событий. Отсутствие ошибок его заменить не может. Сервис здоров, когда регулярно происходит то, ради чего он существует, и молчащий журнал оши-

бок этого никак не гарантирует. К концу главы такой механизм, маленький, встанет и на вашем вечернем проекте.

## Вопрос, на который не было ответа

В июле владелец нашей системы задал вопрос, от которого нельзя отмахнуться: почему ошибки всплывают снова и снова, хотя мы постоянно их чиним?

Вопрос был неудобным, потому что упрёка в нём не было. Чинили действительно постоянно. Каждая поломка разбиралась до причины, по каждой появлялось правило. И тем не менее раз в несколько дней всплывало что-то, что жило сломанным давно.

Ответ нашёлся, когда мы выложили случаи одного месяца в ряд и посмотрели на них как на коллекцию. Три из них в прошлой главе прошли одной строкой; здесь они по порядку и с причинами.

Копии данных не делались трое суток. Причина стоит того, чтобы её рассказать: в скриптах уборка старых копий стояла после загрузки новой. Хранилище заполнилось до квоты, загрузка стала падать, а до строки уборки дело уже не доходило – и каждый следующий запуск упирался в ту же стену. Самоподдерживающийся цикл отказов: поломка сама поддерживала условие, из-за которого не могла исправиться.

Сборщик заявок стоял пустым три месяца. Если история звучит знакомо – да, это тот самый мёртвый мост из прошлой главы. Там он был поломкой, которую нашли и починили; здесь он интересен другим: он умирал молча, с апреля, и ни один сигнал за три месяца не поднялся.

Служба напоминаний пять суток копила зависшие процессы. Каждые пять минут запускался новый, ни один не завершался, и к концу пятых суток они держали девяносто восемь соединений базы

данных из ста. Всё остальное на этой базе начало отваливаться – вот тогда и заметили.

Очередь материалов на ручную проверку старела. Самая старая запись дождалась двадцати пяти дней.

Теперь общее. Во всех четырёх случаях не было ни одной ошибки. Ни строки в журналах, ни красного сигнала, ни упавшей задачи, которая бы об этом сообщила. Каждый случай обнаружил человек – заметил, заподозрил, спросил. Система обо всех четырёх молчала, потому что с её точки зрения ничего плохого не происходило. Просто ничего не происходило.

Мы чинили то, что кричало. А основной вид отказа не кричит.

Отсюда тезис, на котором стоит глава: тишина – это информация, которую никто не подписался получать.

## Сигнал на отсутствие ожидаемого

Сигналы тревоги, которые ставят все, устроены одинаково: случилось плохое – сообщи. Упала задача, кончилось место, сервер ответил ошибкой. Такие сигналы нужны, и у нас они были. Но все четыре истории выше прошли сквозь них, как сквозь открытую дверь: плохого события не случалось ни в одной.

Нужен сигнал второго типа: хорошее не случилось за отведённое время – сообщи. Копия данных не появилась за сутки. Заявка не переслана за час. Журнал задачи не обновлялся полчаса. Первый тип ловит аварии. Второй ловит тихую смерть.

Через неделю после вопроса владельца у нас заработал сторож тишины – небольшая программа, которая раз в шесть часов проходит по списку ожидаемых событий и проверяет, что каждое случалось недавно. Проверок в нём семь, и про этот список важно одно: ни одна проверка не придумана из головы. Свежесть копий данных с

порогом двадцать шесть часов – из истории про трое суток. Число зависших соединений базы с порогом семьдесят – из истории про службу напоминаний. Заполнение диска – из отдельного ожога той же недели. Остальные четыре устроены так же: сначала ожог, потом датчик. Например, свежесть журналов задач по расписанию с порогом полчаса: если задача, которая должна отработать каждые пять минут, полчаса не оставила следа, она мертва. Сторож – это дневник наших поломок, переписанный в форму, которая дежурит.

Два свойства этой программы важнее самого списка, потому что без них сторож быстро начинает вредить.

Первое – повторный сигнал о той же беде подавляется на сутки. Без этого одна поломка присылает одинаковое сообщение каждые шесть часов, к третьему дню его перестают читать, к пятому – читать перестают всё, что приходит с этого адреса. Сигнал, на который перестали реагировать, хуже отсутствия сигнала: он создаёт ощущение, что наблюдение есть.

Второе – пороги взяты из ожогов. Двадцать шесть часов для копий означают «раз в сутки плюс запас на длинную ночную работу»; желание поставить построже, на всякий случай, приводит к ложным тревогам, а те – к предыдущему пункту.

На вашем вечернем проекте ожидаемое событие одно: обращения приходят. Сторож для него – вот он целиком, файл `storozh.py` рядом с проектом:

```
import os
import sqlite3
import sys
from datetime import datetime, timedelta

BAZA = os.environ.get("ZAYAVKI_DB", "zayavki.db")
POROG_CHASOV = 24

try:
    db = sqlite3.connect(BAZA)
    poslednee = db.execute("SELECT MAX(sozdano) FROM
obrasheniya").fetchone()[0]
except sqlite3.Error as oshibka:
    print(f"ТИШИНА: база недоступна ({oshibka})")
    sys.exit(1)

if poslednee is None:
    print("ТИШИНА: обращений не было ни разу")
    sys.exit(1)

if datetime.fromisoformat(poslednee) < datetime.now() -
timedelta(hours=POROG_CHASOV):
    print(f"ТИШИНА: последнее обращение {poslednee}, порог
{POROG_CHASOV} ч")
    sys.exit(1)

print(f"Порядок: последнее обращение {poslednee}")
```

Обратите внимание на первый блок: если база недоступна или в ней нет нужной таблицы, сторож тоже кричит. Это сознательное решение, и оно из разряда тех, что отличают сторожа от украшения: любая беда самого сторожа должна превращаться в сигнал, потому что молчащий по техническим причинам сторож неотличим от довольного.

Запускается раз в день – планировщиком задач вашей системы или, для начала, рукой вместе с утренним кофе. Слово «ТИШИНА» и ненулевой код выхода – сигнал; код выхода это число, которым программа сообщает системе, всё ли прошло гладко: ноль значит порядок, любое другое число – беда. Как привязать к нему письмо

или сообщение в мессенджер, зависит от вашей машины, рецепты собраны на странице книги (notevibe.ru/kniga.html); для вечернего проекта достаточно и привычки запускать.

Порог берите по своему потоку: при одном обращении в неделю суточный порог будет кричать шесть дней из семи, и вы отключите сторожа на третий день.

## Вход важнее выхода

Теперь случай, который дороже всех остальных, потому что его не ловит даже правильно поставленный сторож, если тот смотрит не туда.

Наш конвейер статей – производство: генератор предлагает темы, писатель превращает их в тексты, тексты проходят проверку и очередью уходят на сайты. Наблюдение за ним было, и оно было осмысленным: если публикации остановятся, сигнал поднимется. Публикации не останавливались. Конвейер по всем показателям был жив.

При этом генератор тем месяцами работал вхолостую. В его настройке стоял потолок, из-за которого он предлагал по горстке тем на всю сеть сайтов, темы кончались, писателю было нечего брать. Производство умерло на входе. А выход этого не показывал, потому что между входом и выходом лежал запас готовых статей, и очередь публикации продолжала мерно его тратить.

Заметил человек. Спросил, почему новых материалов давно нет, – при зелёном наблюдении и работающей очереди.

У всякого конвейера есть буфер – запас между производством и выдачей. Буфер нужен и полезен: он сглаживает неровности. Но он же маскирует смерть входа – на столько времени, на сколько его хватает. Датчик на выходе показывает состояние буфера. О произ-

водстве он молчит. Чем больше запас, тем дольше система выглядит здоровой после того, как уже умерла.

После этого случая у нашего конвейера появились датчики входа: ноль новых тем за трое суток – тревога; запас тем меньше пяти – предупреждение; ноль написанных текстов за двое суток – тревога. Выход мы тоже мерим, но больше не считаем его ответом на вопрос «живо ли производство».

Начните с вопроса, датчик потом: что в вашем сервисе вход, а что выход? У сервиса обращений вход – люди, которые доходят до формы, выход – обработанные обращения. Если смотреть только на выход, легко месяц радоваться тому, что необработанных нет, – при потоке, который давно иссяк. Практическое правило простое: держите оба числа рядом, на одном экране – сколько пришло за неделю и сколько обработано. Одна строка, где эти числа стоят рядом; двумя отчётами в разных местах она не заменяется. Расхождение чисел в любую сторону – самый дешёвый диагностический прибор из существующих.

## Кто сторожит сторожа

Остался вопрос, который вы, возможно, уже задали сами: а если сторож умрёт?

Вопрос законный: у сторожа есть расписание, которое можно снести, и окружение, которое можно сломать. Если он умрёт, наступит та самая тишина, которую он сторожил. Молчание наблюдения снаружи неотличимо от здоровья: либо всё хорошо, либо никто больше не смотрит.

Решение выглядит парадоксально, но оно единственное: сторож должен иногда подавать голос без повода. Наш присылает раз в неделю, по понедельникам, одно сообщение – «монитор жив, проблем нет». Пятьдесят две строки в год, цена смешная. Зато отсут-

ствие понедельничного сообщения – само по себе сигнал, и ловит он ровно один сценарий: смерть наблюдения. Всё остальное время сторож молчит: голос без повода, звучащий чаще, чем раз в неделю, быстро становится шумом, а шум перестают слышать.

Здесь наш порядок годится вам без изменений: пусть сторож раз в неделю сообщает, что он жив. Если в понедельник сообщения нет – утро начинается с него.

## Слепая зона в отчёте

Наблюдать приходится за двумя вещами сразу: за системой и за тем, как она о себе рассказывает. Вторая ломается по-своему.

На рабочем экране нашего проекта материалы были видны в двух состояниях: «на проверке» и «опубликовано». Логично и чисто. В июле выяснилось, что сто десять готовых статей не видны нигде: они прошли проверку – значит, уже не «на проверке», и ждали своей очереди публикации – значит, ещё не «опубликовано». Промежуточное состояние, в котором вещь проводила недели, выпало из отчёта целиком.

Вопрос при этом задали тот же, что и в истории с генератором тем: почему нет новых статей. Только причина на этот раз оказалась в отчёте – система работала, очередь была полна, все датчики зелёные. Полезное совпадение: один и тот же симптом дважды за месяц привёл к разным причинам, и оба раза его принёс человек.

Правило из этого случая формулируется жёстче, чем кажется на первый взгляд: у каждого состояния, в котором вещь может находиться дольше минуты, должно быть место в отчёте. Состояние, которое нигде не показывается, эквивалентно потерянному – для всех, кто смотрит на отчёт, этих ста десяти статей не существовало.

Пересчитайте состояния своих обращений. «Новое» и «обработано» – это то, что мы сделали в части 0. А те, что взяли в работу и не

закрыли? Позвонили, договорились перезвонить, ждём ответа человека? Если такого состояния нет в вашем списке – оно всё равно существует, просто в невидимом виде: в голове, в переписке, нигде. Всё, что живёт в невидимом состоянии, рано или поздно теряется, и узнаёте вы об этом от того, кому не перезвонили.

## Что из этого следует

Правило, которым глава закрывается. Отсутствие наблюдения оплачивается каждый раз заново – и, как правило, в худший момент, потому что тихие отказы вскрываются тогда, когда накопились, а это всегда позже, чем они случились. Нам за один июль этот счёт приходил четырежды.

> **Если вы заказчик.** Спросите подрядчика: «Покажите список событий, за отсутствием которых следит система, и порог по каждому». Ответ «у нас настроен мониторинг» сам по себе не значит почти ничего – в девяти случаях из десяти это сигналы об ошибках, сквозь которые тихие отказы проходят свободно. Второй вопрос ещё показательнее: «Что приходит мне или вам, когда всё в порядке?» Если ответ «ничего», то смерть наблюдения никто не заметит.

> **Упражнение.** Поставьте сторожа из этой главы на свой вечерний проект. Потом проверьте его – единственным способом, который считается: устройте тишину нарочно. Переименуйте на время файл базы или подставьте сторожу пустую – и убедитесь, что он закричал. Это то же правило, что и в прошлой главе: проверка, которая ни разу не краснела у вас на глазах, пока лишь надежда, и называть её проверкой рано.

Мы прошли путь от «зелёное может врать» до «тишина – тоже данные». Следующая глава – о самом нервном моменте эксплуатации: как менять работающий проект, не разрушая его. Там снова встре-

тятся и границы правки из первой части, и сторожа из этой – потому что самое опасное время для любого сервиса начинается со слов «сейчас я быстренько поправлю».

## Сейчас я быстренько поправлю

Собрать проект трудно, но безопасно. У нового проекта нет пользователей, в базе нет ничего, что жалко, и худшее, что может случиться, – потерянный вечер. Вся первая половина книги прошла в этом счастливом режиме, где любую ошибку лечит перезапуск.

Менять работающий проект – другая работа с другой ценой. За каждой правкой теперь стоят люди, которые уже пользуются, и данные, которые уже накоплены. Ошибка перестаёт стоить вечер: она стоит чьё-то потерянное обращение, чьё-то испорченное впечатление, чей-то уход. При этом менять придётся постоянно – работающий проект без изменений не живёт, он либо развивается, либо тихо устаревает.

Вы уже умеете собирать, проверять и наблюдать. Осталась последняя дисциплина эксплуатации: вносить изменения так, чтобы после каждого проект оставался целым.

Начнём с тезиса, который переворачивает бытовую интуицию. Кажется, что опасность правки пропорциональна её размеру: большую страшно, маленькую можно «быстренько». Наш опыт показывает другое. Опасность правки меряется числом мест, которые она затрагивает без вашего ведома. Однострочная правка с тремя невидимыми соседями опаснее переписанного модуля, у которого соседей нет. Слова «сейчас я быстренько поправлю» потому и открывают самые дорогие вечера, что произносятся перед правками, у которых соседей не искали.

## Одна правка – одна мысль

Правило вам уже знакомо по первому вечеру: одна задача, одна правка, одна проверка, один коммит. Пришло время объяснить, почему с агентами оно перестаёт быть просто хорошей привычкой и становится техникой безопасности.

Агент по своей природе достраивает. Попросите его поправить статус обращения – и вместе с правкой вы рискуете получить переименованные «для ясности» переменные, переставленные «по логике» функции и обновлённое «для единообразия» оформление. Каждое из этих действий по отдельности осмысленно. Все вместе они превращают вашу правку в изменение, где нужное не отличить от привнесённого. Если после такого изменения что-то сломается, вы не будете знать, которая из пяти непрошеных доработок виновата, – и разбирать придётся все.

Лекарство двухслойное. Первый слой – граница в самой задаче, вы её уже ставили: «меняй только то, что относится к статусу; форму и сохранение не трогай». Второй слой – проверка, которая охраняет запрещённое к изменению: тест из части o защищает сохранение обращений независимо от того, что вы просили. Если после правки статуса он покраснел, агент вышел за границу, и вы узнали об этом за минуту, по горячим следам. Без теста вы узнали бы через неделю, когда след остыл.

Мера правильного размера правки простая: она должна помещаться в одно предложение задачи. «Добавь кнопку обработано» – помещается. «Добавь кнопку, поправь список и заодно наведи порядок в стилях» – три задачи, а слово «заодно» в постановке задачи стоит запретить себе совсем. Именно на «заодно» агент отвечает самой широкой правкой.

## Правка сделана – ещё не значит применена

Теперь ярус, о котором не думает почти никто, и который в июле стоил нам дважды.

В голове у начинающего изменение устроено просто: я поправил файл, значит, всё изменилось. На деле между «я изменил файл» и «люди видят изменение» лежат четыре рубежа, и правка может застрять на любом из них, не подавая признаков застревания.

Рубеж первый: файл изменён на диске. Это единственный рубеж, который виден в редакторе.

Рубеж второй: изменение попало туда, откуда на самом деле работает служба. Если проект живёт на сервере, собирается перед выкладкой или запущен в контейнере – той самой коробке из главы о среде, со своей копией файлов, – то файл на вашем диске и файл, который исполняется, это разные файлы, и между ними есть шаг доставки.

Рубеж третий: работающий процесс перечитал код. Программа, запущенная вчера, держит в памяти вчерашний код и знать не знает, что файл под ней изменился.

Рубеж четвёртый: посетитель получил новую версию. Браузеры запоминают загруженное, и если страница не умеет сообщать «файл обновился», человеку из кеша отдаётся старое.

Оба наших июльских случая – про третий и четвёртый рубежи, и оба поучительны именно будничностью.

Первый. Важная правка легла в файлы, файл внутри контейнера обновился – первые два рубежа пройдены чисто. А служба к тому моменту работала сорок три часа без перезапуска и продолжала исполнять код, который держала в памяти. Формально работа сделана: файл везде новый. Фактически не изменилось ничего, и не изменилось бы до следующего перезапуска – через день, через неде-

лю, когда придётся. Урок мы сформулировали для себя жёстко: проверять надо работающий процесс, диск тут ничего не доказывает. У запущенной службы можно спросить, какой код она исполняет, – и только этот ответ считается.

Второй случай вы уже встречали в главе про врущую проверку, теперь посмотрим на него со стороны правки. Сценарий отправки форм подключался на страницах без версии в адресе. Это значит: даже когда починка доехала до сервера и до работающей службы, часть посетителей ещё долго получала бы из кеша браузера старый сломанный вариант. Для них починка не состоялась. Четвёртый рубеж коварнее прочих тем, что он у каждого посетителя свой: у одних новое, у других старое, и жалобы выглядят мистикой – «у меня работает, у клиента нет».

В вечернем проекте рубежей меньше, но они есть. После правки кода – перезапуск сервера, иначе исполняется прежний. После правки страницы – обновление в браузере с очисткой кеша. Файл и собственная уверенность в свидетели не годятся.

## У правки есть соседи

Третий ярус – о правках, которые прошли все четыре рубежа и всё равно ударили. Потому что изменение приехало, но не везде, где должно было. У правки были соседи, и о них не подумали.

Соседи бывают трёх видов.

**Общий ресурс.** У нас два сборщика – два независимых инструмента, каждый собирает свою часть сайтов – писали таблицу стилей в один и тот же файл. Годами это никому не мешало: правки не пересекались, и никто вообще не знал, что ресурс общий. Потом пересеклись. Кто выложил последним, тот и прав: сто тридцать шесть статей вышли на сайт без оформления, голым текстом. Заметил владелец, по скриншоту. Правило из этой истории: у каждого

файла, который пишут двое, однажды случится этот день – вопрос только в дате. Общие ресурсы надо знать в лицо до того, как они напомнят о себе.

**Забытая точка.** Направление сняли с продажи. Аккуратно, по списку: поправили сайты, остановили генерацию статей на эту тему, перенастроили маршрутизацию. А призыв к действию, который автоматически подставляется в конец каждого видеоролика, жил отдельной записью в настройках бренда – и остался прежним. Ролики продолжали обещать зрителям услугу, которой больше не существовало. Обнаружил владелец, на готовом ролике, глазами. Правило: у решения «мы больше этого не делаем» столько адресов, сколько мест об этом когда-либо обещало. Список адресов составляется до правки. После – его составляет уже клиент, задавая неудобные вопросы.

**Мина в стороне.** Разбирая тот же случай, мы нашли в дереве сборки постороннюю таблицу стилей, пролежавшую там с мая: при точечной выкладке она спала, при полной сломала бы оформление всего сайта. Правило: перед выкладкой целиком посмотреть, что в дереве вообще лежит.

Для вечернего проекта всё это сжимается в один вопрос, который задаётся до правки: где ещё живёт то, что я меняю? Название услуги, цена, телефон, текст письма, адрес – если сущность упомянута в трёх местах, правка в одном создаёт расхождение. Расхождение не подаёт сигналов. Его находит клиент, сверив два места между собой, и находит всегда в неудачный момент.

## Чужие руки, включая свои

Этот ярус легко пропустить, работая одному: конфликтовать вроде не с кем. Но у вас есть агент, и он правит те же файлы, иногда буквально в те же минуты. Две сессии агента в двух окнах – это уже

двое работающих над одним проектом, с классическими командными граблями.

Простой вариант вы уже знаете: в первом вечере мы упоминали, как две сессии, правившие одну папку, затёрли работу друг друга, и на сайт уехал не тот вариант страницы. Ни одна из них не сделала ничего дурного – просто они не знали друг о друге, а общий файл знал обоих.

Второй наш случай тоньше, и его урок дороже. Правку установили и сразу проверили: работает, всё на месте. Через полчаса служба отдавала старые данные, каталог схлопнулся с четырёх тысяч записей до четырёхсот. Что произошло: параллельная сессия сохранила свой вариант файла поверх – взяв за основу копию, снятую до нашей правки. Наша работа была жива в момент проверки и умерла через двадцать минут после неё.

Отсюда два правила, оба выстраданные.

Проверка сразу после установки необходима и недостаточна. Если над проектом работает кто-то ещё – вторая сессия, агент в фоне, коллега, – нужна перепроверка через время. Мы теперь возвращаемся к важным правкам через полчаса-час и смотрим, на месте ли они.

И тут вспоминается сторож из предыдущей главы, потому что перепроверка через час – работа, которую нельзя поручать памяти. Память занята следующей задачей. Если правка важная, а рядом работают чужие руки, самое надёжное – сделать её предметом наблюдения: проверка, которая раз в час убеждается, что нужное значение на месте, стоит десять строк и не забывает. Это тот же приём, что и сигнал на отсутствие ожидаемого, только ожидаемое здесь – ваша собственная работа.

Копии, оставленные чужой работой, – архив, пока не доказано обратное. Когда чья-то версия затёрла чью-то, единственным носите-

лем проигравшей работы часто оказывается резервная копия, которую сделала затёршая сторона. Удалить «лишние файлы» до разбора – значит уничтожить то единственное, из чего затёртое можно восстановить.

Для вечернего проекта защита от всего яруса уже стоит – это git, но с одной оговоркой: он защищает только зафиксированное. Правка, не попавшая в коммит, существует на птичьих правах, и любая параллельная рука – агент, вторая сессия, вы сами завтрашний – может её молча переписать. Частые коммиты из привычки аккуратности превращаются здесь в единственную настоящую страховку.

## Порядок безопасной правки

Соберём главу в последовательность. Она выглядит длинной, но на деле занимает минуты – и за каждым шагом стоит конкретный случай; часть вы только что читали.

1. **Понять, что меняешь и где ещё оно живёт.** Вопрос про соседей задаётся до правки.
2. **Зафиксировать рабочее состояние.** Коммит перед правкой – десять секунд, и у вас есть точка возврата.
3. **Прогнать вхолостую, если инструмент умеет.** Многие инструменты позволяют посмотреть, что изменение сделает, до того как оно это сделает. У нас правило «сначала вхолостую, потом всерьёз» действует во всех рабочих цепочках без исключения – однажды такой прогон показал шестьдесят семь строк, которые ушли бы в работу дублями.
4. **Снять с публикации то, что будет пересобрано.** Наш случай: ролики с устаревшим призывом сначала сняли в черновики, потом пересобирали. В обратном порядке в эфир успевает уйти промежуточная версия.
5. **Одна правка – одна мысль.** С границей в задаче агенту.

6. **Проверить поведение работающей службы.** Путём посетителя, с учётом всех четырёх рубежей.
7. **Прогнать защитные проверки.** Тесты, которые охраняют то, что трогать запрещалось.
8. **Зафиксировать результат.** Коммит с внятной подписью.
9. **Перепроверить через время,** если рядом работает кто-то ещё – включая ваши собственные параллельные сессии. Лучше всего эту работу отдать сторожу из предыдущей главы.

И отдельно – про откат, потому что он венчает весь порядок. Возврат к прошлому состоянию должен быть рутинным решением; если он подвиг, значит, его нет. Проверка простая: можете ли вы вернуть вчерашнюю версию одной командой, не вспоминая, что именно меняли? Если для отката надо восстанавливать в памяти список правок, у вас вместо механизма надежда на память.

Устройте себе учебную аварию прямо сейчас, это две минуты. Удалите в вечернем проекте нужную строку кода – любую, без которой тест краснеет, – и убедитесь, что он покраснел. Затем верните всё одной командой: `git checkout -- .` возвращает файлы к последнему коммиту. Прогоните тест снова, увидьте зелёное. Теперь дорога назад известна не в теории, а руками, и в следующий раз, когда правка пойдёт не туда, вспоминать её не придётся.

Закрывающее правило главы: работающий проект меняют маленькими шагами с фиксацией после каждого. Вопреки ощущению, скорость от этого не падает – шаги маленькие, но их много, и они не откатывают друг друга. Падает цена ошибки.

> **Если вы заказчик.** Три вопроса подрядчику, которые за минуту показывают, умеет ли он менять работающее: «Как вы возвращаете предыдущую версию, если после обновления что-то сломалось?», «Сколько времени занимает возврат?», «Что вы делаете перед выкладкой, чтобы не задеть уже

работающее?». Хороший ответ на первый вопрос содержит название механизма, на второй – минуты, на третий – слова «проверка» и «резервная копия». Ответ «у нас всё под контролем» на все три переводится одинаково: возвращаться мы не пробовали.

**Упражнение.** Выпишите на бумагу три места, где живёт ваш телефон, и три, где живёт цена. Сайт, письмо-подтверждение, подпись в почте, карточка в справочнике, договор, страница в соцсети. Теперь сверьте их между собой. У большинства людей, делающих это впервые, находится хотя бы одно расхождение – старый номер, прошлогодняя цена, адрес, который уже никто не читает. Это и есть соседи вашей будущей правки, увиденные заранее: список, который вы только что составили, в следующий раз сэкономит вам разговор с клиентом, нашедшим два разных ответа на один вопрос.

Эта глава закрывает набор навыков обращения с работающим проектом: проверять, наблюдать, менять. Следующая – о том, на чём всё это стоит и о чём вспоминают позже всего: где на самом деле лежит ваш проект. Из нашей практики: однажды выяснилось, что правки целого дня существуют только внутри работающего контейнера, и одна перезагрузка отделяла их от небытия. Про то, как не оказаться в этом положении, – дальше.

## Где на самом деле лежит ваш проект

Начнём с трёх вопросов.

У вас сгорел ноутбук. Что осталось от проекта?

Вы уехали на неделю, сервис упал, перезапустить его нужно другому человеку. Справится?

База данных обнулилась. Откуда вы возьмёте вчерашнее состояние?

Если хотя бы на один вопрос ответа нет, какая-то часть вашего проекта существует в единственном экземпляре. А единственный экземпляр – это вопрос даты: он потеряется, вопрос только когда. Диски умирают, ноутбуки тонут, серверы пересоздаются, и ни одно из этих событий не предупреждает заранее.

Обычный ответ на вопрос «где лежит ваш проект» звучит как «на компьютере» или «на сервере», и он ничего не значит. Настоящий ответ – перечень мест, где проект существует в единственном экземпляре, потому что именно этот перечень определяет, что вы потеряете. Тезис главы, к которому мы придём через три наших июльских случая: проект существует там, откуда его можно восстановить. Всё остальное – место, где вы с ним работаете.

Все три случая произошли с одним проектом за две недели.

## Код живёт в трёх местах

У кода работающего проекта три дома. Папка, где вы его правите. Место, откуда исполняется запущенная служба, – в прошлый раз мы называли его вторым рубежом. И репозиторий – место, откуда код можно восстановить. В здоровом проекте содержимое всех трёх совпадает. Беда в том, что расходятся они молча.

Июль, плановая настройка резервного копирования. Попутная проверка показывает: контейнер, в котором работает сервис, никак не связан с папкой на диске. Совсем. Весь код, включая правки текущего дня, существует только внутри работающего контейнера. Сверка подтверждает худшее: файлы на диске отличаются от файлов в контейнере, а в репозитории не хватает трёхсот восьмидесяти двух файлов.

Переведём на бытовой язык. Работа целого дня жила в одном экземпляре – в самом хрупком месте из всех возможных, внутри запущенного процесса. Пересоздание контейнера стёрло бы её бесследно, и никакая копия базы данных тут не помогла бы: база хранит данные, код она не хранит. Самое неприятное в этой истории – то, что пересоздание в тот же день уже случалось и уже роняло правки. Это было известно. Это считалось неудобством – «после пересоздания надо доложить правки заново», – и никто не сказал вслух: одна команда отделяла работу дня от небытия.

Урок формулируется коротко. Место, где вы правите файлы, и место, откуда работает сервис, – разные места, пока не доказано обратное. Доказательство одно: сравнить содержимое. Ощущение «ну оно же одно и то же» доказательством не является – у нас оно держалось до первой построчной сверки файлов.

У вас всё проще, но устроено так же. Папка проекта, запущенный сервер, репозиторий. Расхождение рождается из одной привычки – править и не фиксировать. Проверка встроена в git и занимает секунду:

```
git status
```

Каждый файл в списке изменённых существует в единственном экземпляре – в вашей папке. Репозиторий о нём не знает или знает устаревшую версию. Пока список не пуст, часть проекта живёт на птичьих правах, и предыдущая глава показывала, кто первым этим пользуется: параллельная рука, включая вашу собственную.

## Перенос обнажает правду

Второй случай отличается от остальных тем, что здесь потеря уже произошла. Не «могла бы» – произошла, и мы разбирали последствия.

Проект переносили на новый сервер. Перенос шёл по репозиторию, с основной ветки, – разумный, казалось бы, способ: репозиторий на то и существует. Вся работа последних недель жила в другой ветке – а перенос по одной ветке не захватывает остальные, а пятьдесят четыре файла не были в репозитории вообще – часть изменена и не зафиксирована, часть никогда туда не добавлялась. В результате на новом сервере не хватило тридцати файлов: программных классов, шаблонов страниц, миграций базы.

Снаружи это выглядело поучительно. Главная страница нового сервера открывалась и отвечала кодом успеха – тем самым 200, который проверял ваш первый тест в части 0. По всем формальным признакам перенос удался. При этом персональные страницы пользователей отдавали ошибку, а вход в личный кабинет молча не работал: среди потерянного оказался файл с правилами безопасности, без которого страница не могла выполнить ни одну кнопку. Человек нажимал «Войти» – и ничего не происходило. Без ошибки, без сообщения. Ничего.

Узнаёте рисунок? Это врущая проверка из начала части, в новой роли. Открывшаяся главная ничего не говорила о недостающих файлах, потому что главная их не использует. Проверка проходила по дороге, которая потерь не задевала, – и уверенно показывала зелёное.

Разобравшись, мы сформулировали для себя правило. Признак завершённого переноса один: списки файлов сверены построчно, старый сервер против нового. Открывшаяся главная таким признаком не является. Сверка заняла бы минуты. Разбор её отсутствия занял день.

И последняя деталь этой истории, самая неприятная из всех. Когда стали выяснять, где же лежит полный актуальный код, ответ оказался таким: в папке на рабочем компьютере. Вне репозитория.

Единственное место в мире, где проект существовал целиком, – рабочий стол одного ноутбука. Любое восстановление сервера по репозиторию теряло бы всё снова, сколько ни повторяй. Это положение встречается куда чаще, чем принято признаваться, у него даже есть узнаваемая формула: «у меня всё локально, надо бы закоммитить».

Проверьте у себя одним вопросом, он же тест на выживание проекта: если развернуть проект с нуля из репозитория на чистой машине, он заработает? Всё, чего в репозитории нет, при таком развёртывании не появится: незафиксированный файл, настройка, которую вы делали руками и не записали, данные. Отвечать рассуждением бесполезно, мы только что видели, чего стоят рассуждения. Отвечает попытка – и она вынесена в упражнение этой главы.

## Данные, которых нет нигде

Третий случай замыкает лестницу. Проверка резервного копирования на том же проекте дала результат короче некуда: копий нет. Расписание пусто. Каталога, куда они могли бы складываться, не существует. Восемнадцать с половиной тысяч профилей людей и пятьдесят тысяч загруженных ими работ существовали в одном экземпляре, на одном диске, на одном сервере.

Была и предыстория: копия на старом сервере существовала. Её удалили после переезда – переезд же признан успешным, зачем хранить старое. Логика безупречна до того момента, когда вспоминаешь, что на новом сервере копий ещё нет. Сутки данные пролежали без единой страховки.

Здесь черта из главы о данных проходит по живому. Код можно написать заново: дорого, долго, обидно, но можно, и с агентом быстрее, чем когда-либо. Данные написать заново нельзя. Пятьдесят тысяч чужих работ не восстановит ни агент, ни бюджет, ни раскаяние. Поэтому порядок приоритетов обратный привычному: первым

делом страхуются данные, и только потом всё остальное. Привычный порядок обычно другой, потому что код – это «моя работа», а данные – «просто база».

Что мы настроили: ежедневная копия базы в отдельное закрытое хранилище, изолированное от всего публичного, со сроками хранения. И одна деталь, ради которой стоило рассказывать первую историю: код для копии каждый раз берётся из работающего контейнера; диск для этого не годится. Он уже отставал однажды и может отстать снова, а копия обязана снимать то, что работает.

На вечернем проекте это одна команда: скопировать файл базы в другое место, и одну строку в расписании, чтобы это происходило само. С одной оговоркой, которую нельзя опустить: копия на том же диске защищает от «я случайно удалил» и бессильна против «диск умер». Против второго помогает только другой носитель – другая машина, облачная папка, что угодно физически отдельное.

## Копия, из которой не восстанавливали

Остался последний шаг лестницы, и вы, прочитав три главы этой части, наверняка сделаете его сами. Копия есть. А восстановиться из неё можно?

Копия существует ради восстановления, а не ради строчки «настроено». Между «архив создаётся каждую ночь» и «из архива можно подняться» лежит целый список возможных сюрпризов: файл пишется битым, выгрузка обрывается на середине и никто не замечает, у архива нет прав на чтение, пароль от шифрования знал уволившийся, версия базы на новой машине не читает старый формат. Общее свойство у этих сюрпризов одно: все они обнаруживаются в момент, когда восстановление уже понадобилось. То есть в худший из возможных.

Мы проверяли свою первую копию так: подняли её в отдельную временную базу и пересчитали содержимое против работающей. Все восемнадцать с половиной тысяч профилей и все пятьдесят тысяч работ на месте, ноль расхождений. Двадцать минут работы. До этих двадцати минут у нас была надежда; после – знание. Нигде больше в этой книге такой обмен не стоит так дешево.

Копии, к слову, умеют и переставать делаться – молча, как всё в этой части книги; наш случай с этим уже рассказан в главе про тишину. Поэтому полный комплект страховки данных состоит из трёх предметов: сама копия, сигнал на её отсутствие и проверка восстановлением.

У вас это ежемесячный ритуал на десять минут: взять свежую копию, развернуть в отдельную папку, запустить проект на ней, увидеть свои данные. Если этот ритуал не проводится, у вас хранятся файлы. Копиями их делает только проверка.

## Правило, закрывающее часть

Оно короткое: проект существует там, откуда его можно восстановить. Всё остальное, как бы дорого оно ни ощущалось, – рабочее место.

> **Если вы заказчик.** Три вопроса до оплаты, а не после. «Где лежит код и есть ли у меня к нему доступ?» «Как часто делаются копии данных и где они хранятся?» «Когда последний раз проверяли восстановление?» Первый вопрос важнее двух остальных вместе: если код лежит только у подрядчика, вы владеете не проектом, у вас есть подписка на отношения с подрядчиком. Она заканчивается вместе с отношениями.

> **Упражнение.** Разверните свой вечерний проект с нуля: новая пустая папка, только репозиторий, только инструкция из README. В старую папку не подсматривать. Всё, что придётся

> вспоминать или додумывать руками, – список того, чего в проекте нет; по итогам допишите README и зафиксируйте недостающее. Это упражнение – генеральная репетиция и сгоревшего ноутбука, и передачи проекта другому человеку. Второе нам скоро понадобится.

На этом эксплуатация как навык собрана: вы умеете проверять так, чтобы проверка краснела; слышать тишину; менять, не разрушая; и знаете, где лежит то, чем вы владеете. Последняя часть книги – про день, ради которого всё это копилось: проект уходит из ваших рук. Передача помощнику, подрядчику, покупателю – или себе самому через полгода, что почти то же самое. Список мест, где живёт ваш проект, который вы составили в этой главе, там развернётся в список того, что вы отдаёте.

## Четырнадцать вопросов к работающему проекту

Часть III была про то, что происходит с проектом после запуска. Четыре её главы разбирали четыре способа обмануться: поверить зелёной проверке, не услышать тишину, сломать работающее одной правкой и не знать, где проект лежит.

Ниже – всё, что из этих глав следует, в виде вопросов. Шесть вопросов первого вечера выросли здесь в четырнадцать – и выросли тем же способом, каким рос ваш проект: каждый добавился после настоящей поломки, нашей или вашей учебной. Они собраны в одном месте для употребления: пройдите по ним один раз для своего проекта, отвечая делом. Намерение ответом не считается. Ответ «да, наверное» считается за «нет» – именно он стоял у нас напротив половины этих пунктов в июле, когда мы думали, что держим всё под контролем.

## Проверки

**1. Проверяется ли путь целиком?** От щелчка человека до записи в хранилище и уведомления тому, кто должен отреагировать. Включая участки, которые проверять неудобно.

**2. Проверяется ли там, где живёт настоящий сервис?** Зелёное на предпросмотре, на местной копии и в тестовом контуре ничего не говорит о площадке, куда ходят люди, если она отличается хоть одной настройкой.

**3. Умеет ли проверка краснеть?** Сломайте нарочно то, что она проверяет. Если после умышленной поломки она осталась зелёной, у вас украшение вместо проверки.

**4. Проверяется поведение или текст?** По исходникам нельзя заключить ни что сломано, ни что работает. Отвечает только запуск.

## Наблюдение

**5. Какое событие должно происходить регулярно и кто узнает, если оно перестанет?** Сигналы на ошибку тихую смерть не ловят: ноль событий ошибкой нигде не считается.

**6. Мерите вход или только выход?** У любого конвейера есть буфер, и он маскирует смерть входа ровно на столько, на сколько его хватает.

**7. Как вы узнаете, что само наблюдение работает?** Молчание наблюдения снаружи неотличимо от здоровья. Нужен пульс: раз в неделю сигнал «я жив».

**8. Есть ли у каждого состояния место в отчёте?** Состояние, которое нигде не показывается, эквивалентно потерянному – включая те, в которых вещи задерживаются надолго.

### Изменения

**9. Знаете ли вы, где ещё живёт то, что собираетесь менять?** Цена, телефон, название услуги, текст письма: правка в одном месте из трёх создаёт расхождение, которое найдёт клиент.

**10. Как вы убеждаетесь, что правка применилась?** Четыре рубежа: диск, место, откуда работает служба, память запущенного процесса, кеш браузера посетителя.

**11. Можете вернуться назад одной командой, не вспоминая?** Если для отката надо восстанавливать в памяти список правок, у вас вместо механизма надежда на память.

### Фундамент

**12. В каких местах проект существует в единственном экземпляре?** Это и есть перечень того, что вы потеряете.

**13. Заработает ли он, если развернуть с нуля из репозитория на чистой машине?** Всё, чего в репозитории нет, при таком развёртывании не появится.

**14. Когда вы последний раз восстанавливались из копии?** Копия существует ради восстановления. До первой проверки это файлы, а не копии.

---

Четырнадцать вопросов – не программа на выходные и не формальность перед сдачей. Это то, что отличает работающий сервис от собранного: у собранного всё это отсутствует и до первой аварии никак себя не проявляет.

Если хотя бы половина пунктов осталась без ответа делом – вы в том положении, в котором мы были в июле. Ничего постыдного в нём нет: система работала, сайты открывались, статьи выходили.

Просто половина её здоровья держалась на удаче, и мы об этом не знали.

Дальше – часть IV, про день, когда проект уходит из ваших рук. Ответы на эти четырнадцать вопросов там пригодятся дважды: как то, что вы передаёте, и как то, что вы вправе требовать, когда принимаете чужую работу.

# Часть IV. Передача

## Комплект владельца

Глава о том, где лежит проект, закончилась обещанием: репетиция передачи скоро понадобится. Наступило.

Проект уходит из рук по разным поводам: наняли помощника, отдали поддержку подрядчику, продали сервис, передали дело партнёру. А самый частый повод не выглядит передачей вовсе: вы откладываете проект на полгода и, вернувшись, принимаете его у незнакомца, который когда-то был вами.

Мы в этом положении живём постоянно, и это стоит сказать сразу. Наша система собрана и обслуживается с участием агентов, а рабочая сессия агента конечна: она заканчивается, и продолжает другая, которая не помнит ничего. Каждый вечер проект переходит к исполнителю без памяти о вчерашних решениях и без возможности переспросить. У нас «полгода» наступает ежедневно – и всё, что написано в этой главе, добыто именно там, в режиме, где передача повторяется до тех пор, пока не начнёт получаться.

Во всех перечисленных случаях – помощник, подрядчик, покупатель, вы сами через полгода, наша завтрашняя сессия – передаётся одно и то же, и это не код. Передаётся возможность продолжать без вас. Разницу легко увидеть: можно отдать человеку все файлы до единого – и не передать ничего, если запустить проект умеете только вы, доступы разбросаны по вашим почтам, а половина решений живёт в вашей голове. Файлы у него будут. Проекта – нет.

Эта глава о том, из чего возможность продолжать состоит и как убедиться, что вы её действительно отдали.

## Вы владеете тем, что можете унести

Начнём с неудобного тезиса, который стоит примерить на себя до того, как передавать что-то кому-то.

Вы владеете той частью проекта, которую можете забрать и запустить в другом месте. Всё остальное – доступ к чужой доброй воле. Сервис, который живёт только на площадке подрядчика, код, который лежит только у исполнителя, домен, оформленный на чужое имя, – это не активы. Это отношения. Пока отношения хорошие, разница незаметна. Проверяется она в момент, когда отношения кончаются, – а проверять что-либо в этот момент, как вы уже знаете из части про копии, поздно.

Отсюда и определение комплекта владельца: минимальный набор, с которым проект переживает исчезновение любого из участников, включая вас. Предметов в нём четыре.

## Четыре предмета

**Репозиторий с историей.** Репозиторий целиком, со всеми изменениями от первого дня; архив с файлами «на всякий случай» этого не заменяет. Разница принципиальная. Архив отвечает на вопрос «что есть сейчас». История отвечает на вопросы, которые встанут позже: что менялось перед тем, как сломалось; как это выглядело, когда работало; куда вернуться. В части про изменения мы называли откат решением, доступным одной командой, – так вот, у получателя архива этой команды нет. Ему отдали фотографию проекта вместо проекта.

**Инструкция запуска, проверенная на чужой машине.** README из первого вечера здесь вырастает в полный рост. Требование одно, и оно жёсткое: инструкция считается существующей, когда по ней запустил человек, который не участвовал в разработке, на машине, где проекта никогда не было. Инструкция, работаю-

щая только у автора, – это конспект для себя. У нас это правило выстрадано переносом, о котором вы читали: сервер, развёрнутый по репозиторию, лишился тридцати файлов – они жили вне репозитория, и никакая инструкция их не догоняла. Проверка на чужой машине нужна по той же причине: только там видно, чего в проекте нет.

**Доступы – и на чьё имя они оформлены.** Сервер, домен, почта, хранилище файлов, платёжные сервисы, статистика. По каждому – два вопроса: есть ли доступ у владельца и кто указан хозяином при регистрации. Второй вопрос важнее первого, и его почти никогда не задают. Пароль можно передать за минуту. А вот домен, зарегистрированный на подрядчика, принадлежит подрядчику – что бы ни было написано в переписке, и это самая частая мина в отношениях с исполнителями: сайт ваш, адрес сайта – нет.

**Список известных ограничений.** Письменный. Что работает наполовину, что держится на удаче, что решено не делать и почему, где лежат долги. Этот предмет выглядит наименее обязательным и на деле самый показательный: он отличает передачу от избавления. У любого работающего проекта такой список есть – вопрос только, существует он на бумаге или всплывёт у получателя по одному пункту в месяц, в самые неподходящие моменты. Вы составляли его ещё в части 0, когда записывали границу первой версии; теперь он просто повзрослел вместе с проектом.

Заметьте, чего в комплекте нет. Нет красивой документации на сорок страниц – её никто не прочтёт, а устареет она раньше, чем дочитают. Нет комментариев к каждой строке кода. Нет обещания «я всегда на связи, если что – пишите»: обещание предметом не является, это та самая добрая воля, которая имеет свойство заканчиваться.

> **Что написать в договоре.** Если работу для вас делает подрядчик, три пункта из этой главы вписываются в приложение к договору почти дословно: репозиторий с полной историей передаётся заказчику и пополняется по ходу работы, а не в конце; инструкция запуска проверяется на окружении заказчика до подписания акта; список известных ограничений на дату сдачи прилагается письменно. Мы не юристы, формулировки согласуйте со своим. Но смысл этих трёх пунктов стоит дороже большинства остальных страниц договора: без них через год у вас останется работающий сервис, продолжать который не сможет никто.

> **Тревожный признак.** Если правки в ваш проект идут «руками на сервере» или «прямо в базе», минуя репозиторий, – возможность вернуться тает с каждой такой правкой, и комплект владельца худеет, даже если формально всё передано. Просить так не делать надо с первого дня. С первой аварии будет уже поздно и уже не у кого.

## Как это выглядит у нас

Здесь оговорка, которую мы обязаны сделать. Историй передачи проекта стороннему заказчику в наших журналах нет – свои системы мы пока никому не отдавали, и выдумывать красивую сцену сдачи не будем. Всё, что ниже, – правила ежедневной передачи следующей сессии, о которой шла речь в начале. Постановка у неё жёстче заказчика: заказчик хотя бы может позвонить и спросить.

Выживать в таком режиме нас научили три правила, и все три – прямые родственники комплекта владельца.

Первое: журнал, куда записывается каждое изменение. Что сделано, где, что проверено, как откатить. Следующая сессия начинает с чтения журнала, как ваш преемник начнёт с чтения README и

списка ограничений. День, не записанный в журнал, для следующей сессии не существовал – со всеми последствиями, включая повторную работу и противоречащие правки.

Второе: отметка о занятой задаче. Прежде чем трогать общие файлы, сессия записывает, что и зачем берёт. Мы пришли к этому после истории, которую вы читали в части про изменения: две параллельные сессии, один файл, работа одной молча похоронена другой. У людей эта же отметка называется «предупредить коллег, что я в этом файле».

Третье правило – самое важное и наименее очевидное: инструмент не считается сданным, пока не подключён к расписанию и не имеет сигнала тревоги. Оно появилось после случая, который звучит как загадка: инструмент был написан, проверен, работал безупречно – и не работал вовсе. Разгадка скучная: его написали и проверили руками, а подключить к расписанию забыли. Он существовал и бездействовал одновременно, четыре дня, пока разрыв не заметили. С тех пор «сделано» у нас означает: работает без человека, и если перестанет – кто-то узнает. Примерьте это определение на всё, что вам сдают.

Если переложить три правила на язык комплекта, соответствие получится точным: журнал – это README и список ограничений, растянутые во времени; отметка о занятой задаче – дисциплина репозитория; определение «сделано» – готовность к приёмке. О ней – следующая глава.

## Проверка комплекта

Комплект, как и всё в этой книге, считается существующим после проверки, и проверка здесь одна.

Отдайте его. Не на словах – буквально: попросите другого человека развернуть и запустить проект, пользуясь только тем, что вы пере-

дали. Сами молчите. Каждый вопрос, который он вынужден вам задать, – дыра в комплекте: недописанная строка инструкции, незафиксированный файл, доступ, о котором вы забыли, решение, которое живёт только в вашей голове. Записывайте вопросы – это готовый список доработок.

Если одолжить человека негде, работает отложенная версия: та самая проверка развёртыванием с нуля из предыдущей главы, проведённая через месяц. Через месяц вы и есть другой человек.

> **Упражнение.** Соберите комплект владельца для вечернего проекта: репозиторий, инструкция, перечень доступов (для учебного проекта он короткий, но пусть будет письменным), список ограничений из PLAN.md. Отдайте всё это кому-нибудь – коллеге, приятелю, кому угодно с компьютером – и попросите запустить проект, ничего не объясняя голосом. Место, где человек застрянет, покажет дыру точнее любого аудита. Это упражнение занимает вечер и делает с передачей то же, что первый тест сделал с формой: превращает «должно работать» в «проверено».

Комплект собран и проверен – значит, вам есть что передавать. Следующая глава смотрит на ту же сцену с другой стороны стола: как принимать работу, которую сделали для вас, и как по трём вопросам отличить сданный проект от проекта, с которым вас оставили наедине.

## Приёмка

Теперь другая сторона стола. Работу сделали для вас – подрядчик, помощник, знакомый мастер на все руки, – и наступил день, когда её сдают. На экране всё красиво, исполнитель доволен, вам предлагают принять.

Слово «приёмка» звучит казённо и как будто подразумевает недоверие. Ни то ни другое неверно. Приёмка – это способ узнать, что именно вы получили, пока за это ещё можно спросить. Через месяц те же самые вопросы превратятся из рабочих в неудобные, через полгода – в риторические. Хороший исполнитель приёмки не боится, ему есть что показать; для него это возможность сдать работу один раз вместо бесконечного «а ещё вот тут посмотрите».

Главная новость этой главы для читателя без технической подготовки: чтобы принимать работу, не нужно понимать код. Ни строчки. Нужно уметь спрашивать и знать, как выглядит ответ делом. Этому и посвящена глава.

## Три вопроса и три ответа, которые ответами не являются

Начнём с разговора, который мы рекомендуем провести при любой сдаче. В нём три вопроса, вы все их знаете по части III. Интересны здесь не вопросы – ответы. Есть три типовых ответа, которые звучат успокаивающе и означают ровно противоположное тому, как звучат.

Вопрос первый: «Как проверяется, что заявка с сайта доходит до менеджера?»

Типовой ответ: «Мы всё тестировали».

Перевод: проверок нет. Есть память о том, что однажды, во время разработки, оно работало. Прошедшее время в этом ответе – главное слово: тестировали. С тех пор проект менялся, а каждая правка, как вы знаете из части про изменения, имеет соседей. Ответ делом выглядит иначе: вот тест, вот его запуск при вас, вот он зелёный; а вот мы ломаем сохранение – и он красный. Двухминутная сцена, после которой слова не нужны.

Вопрос второй: «Если заявки перестанут приходить – кто и через сколько узнает?»

Типовой ответ: «Мы следим за системой».

Перевод: сигнала на тишину нет, узнаете вы, от несостоявшегося клиента, в неудачный момент. «Следим» – это про ошибки, а тихие отказы, как вы знаете из главы про тишину, ошибок не порождают. Ответ делом: вот сторож, вот сообщение, которое он прислал в понедельник, вот что придёт, если поток остановится, и вот кому.

Вопрос третий: «Как вернуть предыдущую версию, если после обновления что-то сломается?»

Типовой ответ: «У нас всё под контролем».

Перевод: возвращаться не пробовали. Контроль, у которого нет названия механизма и числа минут, – это самочувствие, а не контроль. Ответ делом: название инструмента, история версий при вас на экране, возврат тестовой правки за названное время.

Общее свойство трёх пустых ответов – в них нет ничего, что можно увидеть. «Тестировали», «следим», «под контролем» – глаголы без предъявляемых предметов. Это и есть признак, по которому уклончивый ответ распознаётся без всякой техники: попросите показать, а не рассказать. Всё, что существует, показывается за минуты.

## Приёмочный лист из четырнадцати пунктов

Три вопроса – это разговор. Для полной приёмки у вас уже есть инструмент серьёзнее: разворот «Четырнадцать вопросов к работающему проекту» в конце части III. Мы собирали их как вопросы владельца к собственному проекту, но у них есть второе применение, дословное: это приёмочный лист. Проект, который сдают вам, обязан отвечать на них так же, как ваш вечерний.

Разворот отвечает на вопрос «что должно быть». Приёмка отвечает на другой: как это выглядит, когда вам показывают. Ниже – по пункту на строку, и ни одна из них не требует от вас чтения кода.

1. **Путь целиком** – при вас отправляют заявку с настоящей страницы и показывают её в базе и в уведомлении.
2. **Проверка на рабочей площадке** – показывают, где она прогонялась, и чем эта площадка отличается от той, куда ходят люди.
3. **Проверка краснеет** – при вас ломают сохранение, при вас проверка краснеет.
4. **Вывод по запуску** – на «работает ли» отвечают прогоном; чтение кода вслух в ответ не принимается.
5. **Сигнал на тишину** – называют событие, порог и адресата, показывают, что придёт.
6. **Вход и выход рядом** – показывают экран, где числа «сколько пришло» и «сколько обработано» стоят рядом.
7. **Пульс наблюдения** – показывают последнее сообщение «всё в порядке» с датой.
8. **Состояния в отчёте** – перечисляют состояния обращения и показывают каждое на экране.
9. **Соседи правки** – показывают список мест, где живут цена, телефон, название услуги.
10. **Правка применилась** – показывают, какой код исполняет работающая служба и что видит посетитель в чистом браузере.
11. **Возврат назад** – называют механизм, показывают историю версий и откатывают тестовую правку при вас за названное время.
12. **Единственные экземпляры** – называют перечень того, что существует в одном месте.

13. **Развёртывание с нуля** – при вас разворачивают проект по инструкции на чистой машине. Самый ёмкий пункт: один прогон закрывает сразу несколько остальных.
14. **Восстановление из копии** – показывают восстановленную копию и дату последней проверки.

Ни один пункт не требует от вас квалификации. Все требуют времени – приёмка по такому листу занимает часы. Сравните с ценой альтернативы: почти все истории этой книги, от кнопки в никуда до тридцати потерянных файлов, были бы пойманы этим листом за один проход. Каждая из них стоила дороже.

И одно предостережение в обратную сторону. Лист – не орудие давления. Проект, прямо отвечающий на десять пунктов из четырнадцати с письменным списком остальных четырёх, – это хорошая, взрослая сдача; требовать идеала на вечернем масштабе бессмысленно и вредно. Отличать надо другое: «этого нет, вот список, вот сроки» от «да всё там нормально». Первое – рабочее состояние. Второе – всё тот же глагол без предмета.

> **Сцена приёмки.** Как это звучит вживую, три реплики.
>
> – Покажите, как заявка доходит до почты. – «Ну, мы всё тестировали, всё доходило». – Отлично, давайте прямо сейчас отправим одну и посмотрим на неё в почте.
>
> – Что будет, если заявки перестанут приходить? – «Мы следим». – Покажите последнее сообщение от системы слежения.
>
> – Как откатиться, если обновление сломает сайт? – «Всё под контролем». – Назовите команду и время.

## Приёмка у самого себя

Теперь случай, ради которого эту главу стоит читать даже тем, кто никогда ничего не закажет.

Проект, к которому вы не прикасались три месяца, – чужой проект. Автор, который его писал, больше не существует: он помнил, почему выбран этот путь, где закопана времянка, что нельзя трогать. Вы – новый человек с его файлами, и отношения с этим проектом надо начинать с приёмки, до всяких правок. Открыть лист из четырнадцати пунктов и пройти без поблажек себе: что разворачивается, что проверяется, что и куда сигналит, откуда восстанавливаться. Обнаруженные дыры – опись имущества, стыдиться тут нечего: теперь вы знаете, чем владеете.

Худшая альтернатива этому часу – привычный сценарий: открыть код, «быстренько поправить» нужное место и запустить цепочку, которой была посвящена целая глава. В чужом проекте – а трёхмесячный ваш именно таков – соседей правки не знает никто.

## Если проект уже принят давно и без всего этого

Частый случай: сервис работает год, сдавали его без листов и вопросов, исполнитель давно занят другим. Что теперь – всё пропало?

Ничего не пропало. Проведите приёмку задним числом, у себя, без исполнителя. Тот же лист, те же «покажите» – только показывать будете себе. На выходе получится список расхождений: чего нет, что не проверяется, куда нет доступа. Этот список – не приговор и не смета на переделку. Это карта, и следующая глава целиком про то, что делать с такой картой: что чинить, что переписывать, а что осознанно отпустить.

> **Упражнение.** Проведите приёмку у самого себя. Возьмите свой самый давний работающий проект – настоящий, вечер-

ний из этой книги не в счёт: сайт, табличку с макросами, бота, что угодно, чем пользуетесь давно и что делали вы или для вас. Пройдите четырнадцать пунктов, отвечая делом. Результат записывайте без прикрас, в две колонки: «показано» и «нет предмета». Вторая колонка – это ваша карта для следующей главы. У нас после первого такого прохода по собственной системе вторая колонка оказалась длиннее первой – с этого, собственно, и началась половина историй этой книги.

Приёмка отвечает на вопрос «что мне сдали». Следующая глава – про вопрос, который встаёт следом и мучает каждого владельца поработавшего проекта: это чинить, переписывать или пора отпустить.

## Чинить, переписывать или отпустить

После приёмки – своей ли работы, чужой ли – у вас на руках карта расхождений: чего нет, что не проверяется, что держится на удаче. И встаёт вопрос, который владельцы поработавших проектов задают чаще любого другого, обычно с тоской в голосе: это вообще лечится? Или проще снести и собрать заново?

Вопрос настоящий, и у него есть плохой способ решения – по ощущению. Ощущение всегда голосует за снос: старое надоело, в новом не будет старых ошибок, а агент соберёт быстро. Оба обещания ложные. Ошибки будут – новые, и вы познакомитесь с ними без карты; а «быстро» относится к генерации, про цену которой без эксплуатации вы прочли целую книгу.

Решение принимается иначе, и мерило у него одно.

## Мерило: воспроизводимость

Считать недостатки бесполезно: их список есть у любого работающего проекта, включая образцовые. Смотреть надо на другое – предсказуемо ли поведение проекта и проверяемы ли последствия правок.

**Чинить**, когда главный сценарий работает; поломки локальны – починка одного не разваливает соседнее; есть история изменений и возможность вернуться; понятно, где что лежит, или это можно выяснить. Такой проект – зрелый организм с болячками. Болячки закрываются по одной, по правилам части про изменения, и каждая закрытая делает следующую проще.

**Переписывать**, когда правки непредсказуемы: тронули одно – сломалось три соседних, и так дважды подряд; истории нет, вернуться некуда; никто, включая автора, не знает, зачем половина файлов; проект невозможно развернуть заново – он работает там, где работает, и это единственный экземпляр. Такой проект уже нельзя менять – можно только надеяться. Важно: переписывать – это заново строить то же самое по правилам этой книги, с частью 0 в качестве первого вечера. Карта расхождений из приёмки при этом превращается из приговора в техническое задание – редкий случай, когда от плохой новости есть прямая польза.

И есть третий исход, о котором почти не говорят, потому что он звучит как поражение. **Отпустить** – когда проект решает задачу, которая больше не стоит. Дело даже не в качестве решения – самой задачи больше нет: направление закрыто, процесс изменился, сервисом давно не пользуются, и единственное, что он производит, – расходы на поддержание и чувство вины. Держать такой проект живым – то же, что чинить лестницу к двери, которую замуровали. Отпустить – значит выключить осознанно, сохранить то, что стоит сохранить, и освободить руки. С «бросить» это не совпадает.

### Как это выглядело у нас

Три решения из наших журналов, по одному на исход. Все три приняты в течение одного месяца – нормальная частота для работающей системы.

**Починили – полумерой, и это было правильно.** История с двумя сборщиками и одним файлом стилей закончилась для читателя в части III, а для нас там было продолжение: развязав сборщики, мы получили те самые сто тридцать шесть статей, которые снова читались, но выглядели чужим оформлением – рамка и подпись другого раздела. Правильное решение по учебнику – немедленно вернуть родной шаблон. Правильное решение по жизни оказалось другим: люди получили работающие страницы в тот же вечер, а родной шаблон записан в долг, письменно и с датой. Отсюда правило, которым мы пользуемся с тех пор: полумера с письменной записью и сроком – инженерное решение; полумера без записи – будущая забытая поломка. Разницу между ними вы уже знаете по немым состояниям из главы о тишине: что не записано, того не существует.

**Выключили – но не удалили.** Один из наших инструментов был признан ненужным: его работу теперь делала другая часть системы, лучше. Остановили. А каталог с его файлами оставили на месте – и через день выяснилось, насколько правильно: внутри каталога жили шаблоны, которыми пользовалась совсем другая, работающая часть системы, о чём не помнил никто. Удаление «мёртвого» каталога уронило бы живое. Правило: выключить и удалить – два разных действия, и между ними должно пройти время. Выключенное можно включить за минуту; удалённое – смотри главу про копии.

**Отпустили направление – и прошли по всем местам, где оно было обещано.** Группа наших сайтов месяцами не приводи-

ла ни одного человека. Решение далось трудно – в сайты был вложен труд, – но числа были красноречивее сентиментов: задача, под которую они строились, не подтвердилась. Направление свернули, полезные материалы перенесли на главный сайт. И вот что оказалось самым трудоёмким: свернуть – значит пройти по всем местам, где направление себя обещало. Вы помните из части про изменения историю с призывом в видеороликах, который продолжал предлагать закрытую услугу: это был хвост ровно этого решения. У решения «мы больше этого не делаем» адресов всегда больше, чем кажется, и последние из них находит взгляд на готовый ролик, там, куда поиск по файлам не заглядывает.

## Дерево решений на одну страницу

Сведём главу в таблицу – по ней решение принимается за вечер, прямой ответ на вопросы у вас уже есть после приёмки.

| Вопрос | Да | Нет |

|---|---|---|

| Задача, которую решает проект, ещё стоит? | дальше | **отпустить** |

| Главный сценарий работает? | дальше | к следующему вопросу |

| Правки предсказуемы – чинится одно, не ломается соседнее? | **чинить** | дальше |

| Есть история изменений и путь назад? | **чинить**, начиная с самых болючих пунктов карты | дальше |

| Проект можно развернуть заново на другой машине? | **чинить**, но первым делом – комплект владельца | **переписывать**, карта расхождений = задание |

Два примечания к таблице. Первое: «переписывать» в нижнем правом углу – это самый дорогой исход, и попадание туда почти всегда означает, что у проекта не было ни репозитория, ни копий, ни комплекта. Всё, чему учила эта книга, – способы никогда не оказаться в этой клетке. Второе: решение «чинить» без письменного списка и сроков само собой превращается в «работает – не трогай». Это четвёртый исход, единственный по-настоящему плохой: он выглядит как решение, а является его отсутствием, и заканчивается всегда одинаково – аварией в момент, который выберет не владелец.

Если после приёмки и таблицы вы всё равно не понимаете, в какой клетке ваш проект, – это нормально: изнутри видно хуже, чем снаружи. Такой вопрос решается взглядом в проект, книга тут уже сделала, что могла. Одно правило для любого такого разбора: отчёт сам по себе ценности не имеет, он зачитывается в работу по исправлению.

> **Если вы заказчик.** Когда подрядчик предлагает «проще переписать с нуля», попросите его заполнить с вами таблицу из этой главы, по вопросу за раз, с предъявлением. Иногда он прав, и таблица это покажет. Но «переписать» – самый удобный для исполнителя ответ: старый код можно не разбирать, смету можно писать заново. Таблица переводит разговор из ощущений в проверяемое – а это, как вы уже знаете, главный ход этой книги в любом споре.

> **Упражнение.** В вашем проекте – настоящем, из упражнения прошлой главы – есть участок, который вы обходите стороной. Все знают свой такой участок: его страшновато трогать, про него говорят «потом разберусь». Проведите его через таблицу и запишите вердикт: чинить – тогда первый шаг и дата; переписывать – тогда что именно и когда; отпустить – тогда что сохранить перед выключением. Одна запись, десять минут. Уча-

> сток, у которого появилась записанная судьба, перестаёт быть источником фонового страха – а фоновый страх, накопленный по всем таким участкам, и есть то чувство «страшно трогать свой проект», с которого начинается путь большинства читателей этой книги.

Дальше – финал книги. Он короткий: о том, что изменилось за эти страницы, что не изменится никогда, и где проходит граница, за которой наша книга заканчивается, а ваш проект продолжается.

## Финал книги

Книга началась с формы, которая говорила «Спасибо» и выбрасывала обращение. Закончим тем, что между той страницей и этой изменилось – и что осталось как было.

### Что изменилось

У вас есть работающий сервис. Небольшой, зато настоящий: с хранением, которое переживает перезапуск, с проверкой, которая краснеет от настоящей поломки, со сторожем, который заметит тишину, с историей изменений, по которой можно вернуться, и с комплектом, который можно отдать другому человеку. В книге не было ни одной строки кода и ни одной команды, которую вы не запускали. Всё, что вы теперь знаете, вы знаете не из чтения. Вы это делали.

Изменилось и кое-что менее осязаемое. В первый вечер убедительная картинка на экране вызывала доверие. Теперь она вызывает вопросы – те самые шесть, потом четырнадцать. Это профессиональное зрение, и с подозрительностью его путают только те, у кого его нет; самая дешёвая из всех страховок: она куплена за один сломанный вечерний проект и десяток наших историй, а страхует от историй, цену которых вы теперь знаете.

## Что не изменилось

Агент по-прежнему достраивает недосказанное, и достраивает правдоподобно. Это его природа. Она не исправится в следующей версии и не лечится удачной формулировкой запроса – она вообще не болезнь. Ровно та же способность, которая заполняет пробелы вашей задачи самыми дешёвыми решениями, пишет за минуты код, на который недавно уходили недели. Инструмент не станет отвечать за результат. Отвечает владелец – и после этой книги слово «владелец» для вас, надеемся, звучит конкретно: тот, у кого репозиторий, копии, сторож и список ограничений.

Не изменилось и главное соотношение, с которого книга начиналась: собрать – быстро, содержать – работа. Генерация закрыла первое. Второе осталось людям, и это надолго.

## Чем написана эта книга

Признание, которое вы, возможно, уже заподозрили: эта книга написана вместе с агентом – по её же собственным правилам. Каждая команда прогонялась на чистом окружении до попадания в текст, и прогон исправно ловил то, чего не увидел бы никакой редактор: пакет, без которого не работают формы; учебную поломку, которая падала эффектно, вместо того чтобы падать поучительно; сторожа, который на пустой базе умер бы с трассировкой вместо внятного сигнала. Главы вычитывала отдельная свежая сессия, потому что сессия, написавшая текст, повторов в нём не видит – она их и породила. А проверка фактов нашла в нашем собственном рабочем каноне ошибочную цифру, которая прожила там неделю и едва не уехала в эту книгу как «документально подтверждённая».

Рассказываем это не ради занятности. Это ответ на вопрос, который висит над каждой страницей любой книги про вайбкодинг: можно ли с агентом делать настоящие вещи? Можно. Ровно тем способом,

который здесь описан: с проверкой запуском, свежим взглядом на вычитке и владельцем, который отвечает за результат. Книга, которую вы дочитываете, – ещё один проект, собранный по этой методике, и вы только что проверили её работоспособность самым надёжным из способов.

## Где заканчивается эта книга

Скажем прямо, чему вы не научились. Проектировать системы под высокую нагрузку. Глубоко читать чужой код. Выбирать между архитектурами. Строить то, что делает команда инженеров. Эта книга про другой навык, который встречается реже перечисленных: не потерять то, что собрал, и отдать то, что сделал, в состоянии, за которое не стыдно.

Отсюда два пути, и оба достойные. Первый: учиться ремеслу глубже – теперь у вас есть фундамент, на котором настоящая инженерия строится, а не повисает. Второй: признать, что ваш проект перерос вечерние силы, и взять инженера – теперь вы умеете его выбирать, принимать его работу и знаете, что должно остаться у вас, когда он уйдёт.

Плохой путь один, и вы знаете его под именем четвёртого исхода: «работает – не трогай». После четырнадцати вопросов оказаться в нём можно только по решению; неведение кончилось, а это другая ответственность.

## Последнее

Если, пройдя книгу, вы посмотрели на свой настоящий проект – тот, что работает и приносит пользу, вечерний не в счёт – и увидели, что половина вопросов остаётся без ответа делом, знайте: это нормальное состояние, мы сами были в нём в июле, с двумя десятками сайтов. Из него есть выход, и он состоит из маленьких шагов с

фиксацией после каждого – вы читали, как это делается, на наших ошибках.

А если захотите, чтобы на ваш проект посмотрели снаружи – вы знаете, где нас найти. Начните с четырнадцати вопросов: с ними разговор о любом проекте становится короче и дешевле.

И мера на будущее, без скидок: если через месяц после этой книги вы не сможете ответить делом хотя бы на половину из четырнадцати вопросов – у вас всё ещё картинка. Просто большая.

Форму из первого вечера, кстати, не выключайте. Пусть принимает обращения. Она заслужила.

---

*Книга написана человеком и ИИ-агентом в четыре руки, по её же собственным правилам: структуру и вычитку вела одна модель, прозу – другая, каждый пример проверялся запуском, ответственность за каждое слово – на авторе-человеке. Список инструментов, которыми она собрана, – на странице книги notevibe.ru/kniga.html, с датой сверки.*