RENAR

Практические руководства по внедрению RENAR

RENAR · Практическое руководство · Версия 1.1 | 19.09.2026

Авторы: Вадим Соглаев, Андрей Юмашев

00. Быстрый старт

> *Сквозной пример: маленький проект «email/password sign-up». Полный цикл RENAR с минимальным набором артефактов: ТЗ → ADAPT → BR → SR → SPEC → TC. Время: ~30 минут чтения + проб на бумаге; ~2-3 часа если делать вживую на носителе. Предпосылки: core/renar-core.md (≤ 10 мин). Полноразмерный пример с 2FA — 01-walkthrough.md. Язык RU-корпуса — standard/00 §0.7.*

Шаги ниже — один из возможных порядков работы, выбранный для маленького примера; нормативны только предусловия и правила стандарта (standard/01 §1.2.1) — порядок ваша организация вправе выстроить по-своему.

После этого старта у вас будет: понимание полного цикла RENAR на одном маленьком примере; готовые YAML-шаблоны для каждого типа артефакта; опыт прохождения двух шлюзов (QG-ADAPT-approve ≡ **QG-3 Architecture Gate** §10.4.1 + **QG-2 Verification Gate**); точка для масштабирования.

Предпосылки

Носитель (`substrate`) — система хранения и версионирования артефактов. RENAR независим от вида хранилища; для быстрого старта подойдёт любой носитель с capabilities V1-V6 (reference/01 §2.7): git с PR-ревью; документо-ориентированное хранилище; БД с историей и подписями.

Структура папок (соглашение по умолчанию для git):

```
my-project.req/
  tz/                   # immutable ТЗ от клиента
  actz/                 # ACTZ — протоколы уточнения ТЗ (двусторонняя подпись)
  adapt/                # ADAPT артефакты
  br/                   # Business Requirements
  sr/                   # System Requirements
  specs/{arch,api,data,ui,sec,ai,proc,int,ops,test,doc}/
  tests/                # TC (tc-type incl. contract)
  at/                   # AT — приёмочные тесты от итогового ТЗ
```

Для других носителей — эквивалентная организация по типам документов или namespace.

Шаг 1. ТЗ (5 мин)

Клиент даёт небольшое ТЗ. Это **договорной неизменяемый** документ — после подписания не редактируется.

`tz/TZ-2026-001.md` (фрагмент):

```
---
id: TZ-2026-001
```

```
title: "Регистрация пользователей через email"
signed-date: "2026-05-15"
signed-by-client: "Иванов И.И., PM, ClientCo"
---

# ТЗ-2026-001

## §1. Контекст
Создать систему регистрации пользователей по email.

## §2. Требования
- Пользователь может зарегистрироваться через email и пароль.
- После регистрации пользователь подтверждает email.
- После подтверждения — доступ в личный кабинет.

## §3. Ограничения
- Только web-приложение.
- Хранение в РФ (ФЗ-152).
```

ТЗ подписано клиентом → неизменяемо. Все правки и интерпретации идут через ADAPT.

Шаг 2. ADAPT `draft` (10 мин)

Создаём `adapt/ADAPT-001-main.md`. AI-агент или инженер заполняет **Forward** (как поняли) и **Backward** (что неясно).

```
---
id: ADAPT-001
title: "Адаптация ТЗ-2026-001 — Регистрация через email"
type: ADAPT
source-tz: { id: TZ-2026-001, signed-date: "2026-05-15", signed-by-client:
"Иванов И.И., PM, ClientCo", document-version-ref: "<идентификатор версии>" }
status: draft
created: "2026-05-16"
ai-provenance: { generated-by: "anthropic-claude-opus-4-7@2026-05-16", prompt-
template: "prompts/adapt-from-tz.md@v1.0", context-tokens: 1024, output-tokens:
2048, human-edits: false }
---
```

Forward §2 Требования:

```
**Цитата:** «Пользователь может зарегистрироваться через email и пароль. После
регистрации пользователь подтверждает email. После подтверждения — доступ в
личный кабинет.»

**Интерпретация:** POST /auth/sign-up с {email, password}. Создаётся User status
`unverified`. Отправляется email с verification link. Клик → status `verified`.
Только `verified` могут войти.
```

```
**Достроенные сценарии:** sign-up; email verification; первый вход.
**Охват:** входит email/password + verification + доступ в ЛК. НЕ входит: OAuth,
SMS, 2FA, сброс пароля.
**Forward links** (auto-populated после утверждения): BR-01, SR-01, SR-02, SR-03.
```

Backward (3 записи):

```
### B-001: gap — попытка входа неверифицированного
Статус: open. Описание: ТЗ не описывает поведение системы при попытке входа до
верификации email.
Вопрос клиенту: показывать сообщение «подтвердите email» с кнопкой повторной
отправки — или 401 без объяснения?

### B-002: regulatory — ФЗ-152 хранение в РФ
Статус: open. Описание: ТЗ §3 требует «хранение в РФ». Что конкретно: дата-центр,
юрисдикция provider, отдельная инсталляция?
Вопрос клиенту: уточнить scope ФЗ-152.

### B-003: gap — повторная отправка email
Статус: open. Описание: что если verification email потерян? rate limit?
Вопрос клиенту: policy на повторную отправку.
```

Шаг 3. ACTZ и ADAPT `approved` (5 мин)

Вопросы к клиенту уходят не «в переписку»: архитектор собирает их в **ACTZ** — протокол уточнения ТЗ (в договорном обороте — «Протокол уточнения ТЗ № 1»). ACTZ подписывают обе стороны, и именно он несёт контрактный вес (standard/07 §7.13). Lifecycle backward: `open → asked-to-client → answered → resolved → frozen (после approval)` ; lifecycle протокола: `draft → sent → signed` .

`actz/ACTZ-001.md` — решения клиента на языке обязательств:

```
---
id: ACTZ-001
title: "Протокол уточнения ТЗ № 1 к TZ-2026-001"
type: ACTZ
tz-ref: TZ-2026-001
resolves: [B-001, B-002, B-003]
status: signed
client-signature: { signed-by: "Иванов И.И.", role: PM, organization: "ClientCo",
signed-at: "2026-05-18T15:30:00Z" }
vendor-signature: { signed-by: "Петров П.П.", role: architect, organization:
"VendorCorp", signed-at: "2026-05-18T15:40:00Z" }
---

## §1. Вход до подтверждения email
```

```
Система показывает сообщение «подтвердите email» и кнопку повторной отправки.
Отказ без объяснения не допускается.

## §2. Хранение данных
Данные хранятся в дата-центре на территории РФ; юрисдикция — РФ. Выбор provider
остаётся за исполнителем.

## §3. Повторная отправка письма
Не чаще 1 раза в 5 минут и не более 5 раз в сутки.
```

Подписанный протокол возвращается в ADAPT: каждая находка получает `decided-in` — ссылку на пункт **подписанного** ACTZ (§7.4.5). Без такой ссылки находка не может стать `resolved`.

```
### B-001: resolved (decided-in: ACTZ-001 §1)
→ добавлен сценарий в Forward §2; создан SR-04.

### B-002: resolved (decided-in: ACTZ-001 §2)
→ data-residency: ["RU"] в Forward §2.

### B-003: resolved (decided-in: ACTZ-001 §3)
→ rate-limit правила в Forward §2; SR-04 уточнён.
```

Все backward → `resolved`, `open-questions-count = 0`. **QG-ADAPT-approve** (≡ QG-3) — все предусловия §10.8.3 выполнены. ADAPT — внутренний документ интерпретации, поэтому его подписывает **только архитектор**; клиентская подпись живёт в ACTZ (§7.5):

```
status: approved
approval:
  # client-signature отсутствует: ADAPT — интерпретация, а не обязательство
  architect-signature: { signed-by: "Петров П.П.", role: architect, signed-at:
"2026-05-18T16:00:00Z" }
ai-provenance: { human-edits: true }    # архитектор отредактировал AI-draft
open-questions-count: 0
resolved-questions-count: 3
```

После утверждения ADAPT **неизменяем** (frozen). Все BR/SR/SPEC ссылаются на approved ADAPT, не на ТЗ напрямую.

Итоговое ТЗ = начальное ТЗ + все подписанные ACTZ (§7.14). Именно против него потом сдаётся система — не против ADAPT.

Шаг 4. BR-01 (3 мин)

`br/BR-01-user-registration.md`:

```
---
id: BR-01
title: "Регистрация пользователей через email"
type: BR
priority: must
source: { adapt: "ADAPT-001", adapt-section: "Forward §2", tz-section: "§2" }
business-context:
  stakeholder: "Иванов И.И. (PM, ClientCo)"
  business-goal: "Дать пользователям возможность создать аккаунт и получить
доступ к продукту"
business-outcome:
  measurement-type: kpi
  kpi-name: "registration-conversion-rate"
  measurement-method: "registered / visited_signup * 100%"
  baseline-value: 0
  target-value: 60
  target-met-by: "2026-09-01"
data-classification: { contains-pii: true, retention-days: 2555, data-residency:
["RU"] }   # 7 лет по ФЗ-152
compliance: [{ standard: "ФЗ-152", article: "ст.6,12" }]
ai-provenance: { generated-by: "anthropic-claude-opus-4-7@2026-05-18", human-
edits: true }
---

# BR-01: Регистрация пользователей через email

Пользователь должен мочь самостоятельно создать аккаунт через email/password
и получить доступ к личному кабинету после подтверждения email.
```

Шаг 5. SR-01..SR-04 (3 мин)

Декомпозируем BR на verifiable SR. Каждый SR ссылается на approved ADAPT.

`sr/SR-01-sign-up.md` (frontmatter + body):

```
---
id: SR-01
title: "Sign-up через email/password"
type: SR
parent: { id: BR-01 }
source: { adapt: "ADAPT-001", adapt-section: "Forward §2", tz-section: "§2" }
constrained-by: ["SPEC-API-01", "SPEC-DATA-01", "SPEC-SEC-01"]
quality-characteristic: [functional-suitability, security]
---

## Описание
POST /auth/sign-up принимает {email, password}.
- Валидные данные → User status `unverified`, отправляется verification email,
201.
```

```
- Невалидный email (regex/format) → 422 с указанием поля.
- Уже занятый email → 409.
- Слабый пароль (< 8 chars или blacklist) → 422.
```

Аналогично — SR-02 (email verification), SR-03 (вход для verified), SR-04 (повторная отправка email с rate limit из B-003).

Шаг 6. SPEC-* (3 мин)

`specs/api/SPEC-API-01-auth.md` (фрагмент):

```
---
id: SPEC-API-01
title: "REST API аутентификации"
type: SPEC-API
source: { adapt: "ADAPT-001", adapt-section: "Forward §2", tz-section: "§2" }
api-style: rest
api-version: "v1.0.0"
versioning-strategy: url-path
authentication: bearer-jwt
rate-limits: [{ endpoint: "POST /auth/resend-email", limit: "1/5min/user;
5/24h/user" }]
contract-file: { format: openapi-3.1, location: "contracts/auth-api.yaml" }
depends-on: ["SPEC-DATA-01", "SPEC-SEC-01"]
referenced-by: ["SR-01", "SR-02", "SR-03", "SR-04"]   # auto-derived
---
```

Аналогично — SPEC-DATA-01 (схема User), SPEC-SEC-01 (auth model + ФЗ-152 controls).

Шаг 7. TC pos/neg парность (5 мин)

Каждый SR — минимум 1 позитивный + 1 негативный TC. Каноничная схема — reference/02 §8.

`tests/TC-01-signup-success.md` (позитивный для SR-01):

```
---
id: TC-01
title: "Sign-up: успешная регистрация"
type: TC
tc-type: system
verifies: [{ id: SR-01 }]
negative: false
automation: { status: automated, set-version: "1.0", location:
"tests/auth/test_signup.py::test_signup_success", runner: pytest }
---

## Given
```

```
- новый email (uniquely-generated@test.com); валидный password (длина ≥ 8, не в
blacklist).

## When
POST /auth/sign-up {email, password}

## Then
- status 201; body {"user_id": "<uuid>", "status": "unverified"}; User в БД с
status `unverified`; verification email отправлен (mock).
```

`tests/TC-02-signup-invalid-email.md` (негативный для SR-01):

```
---
id: TC-02
title: "Sign-up: отклонить невалидный email"
type: TC
tc-type: system
verifies: [{ id: SR-01 }]
negative: true
automation: { status: automated, set-version: "1.0", location:
"tests/auth/test_signup.py::test_signup_invalid_email", runner: pytest }
---

## Given
- email "not-an-email" (без `@`); любой валидный password.

## When
POST /auth/sign-up {email, password}

## Then
- status 422; body {"field": "email", "error": "invalid format"}; User в БД НЕ
создан; email НЕ отправлен.
```

Аналогично пары для SR-02, SR-03, SR-04. Для SR-04 (с rate limit) — дополнительный TC, проверяющий блок после 5-й попытки за 24 часа.

Шаг 8. Утвердить версию комплекта, запустить TC, записать верификацию (2 мин)

8.1 QG-0 комплекта. BR-01, SR-01..04, три SPEC и все TC-нормы лежат в черновике комплекта. Собственного статуса у них нет (standard/10 §10.7) — утверждается комплект целиком: на каждое утверждение есть пара TC-норм, состязательный обзор пройден, Архитектор подписывает. Выпускается запись версии (§10.5.4):

```
# renar/VERSIONS/1.0.md
set-version: "1.0"
major: false
```

```
approved-by: "Иванов И.И."
approved-at: "2026-05-19T09:00:00Z"
resolves-to: "<нативный для носителя указатель на снимок>"
changes: { added: [BR-01, SR-01, SR-02, SR-03, SR-04, SPEC-API-01, SPEC-DATA-01,
SPEC-SEC-01, TC-01, TC-02, "…"] }
```

8.2 QG-1 и прогон. Реализации TC написаны против версии `1.0` (`automation.set-version: "1.0"`), допущены к прогону; test runner на носителе запускает их и обновляет `last-run` :

```
# tests/TC-01-signup-success.md (после прогона):
last-run: { date: "2026-05-19T10:00:00Z", result: pass, runner-id: "test-
runner@1.0", run-ref: "<ссылка>", set-version: "1.0" }
```

8.3 QG-2 версии. Когда **все** TC версии `1.0` зелёные при `last-run.set-version = 1.0` и у каждого записан переход «красный → зелёный» (standard/09 §9.18.2) → спот-чек инженера (§9.14): выборка из 5 прошедших TC, наводимая машинными сигналами (в первую очередь — TC без красной истории), сверяется с SR. Если спот-чек пройден → **QG-2 Verification Gate** пройден → runner дописывает запись верификации в запись версии:

```
# renar/VERSIONS/1.0.md (дополняет только runner):
verification:
  - { product-version: "<указатель на сборку>", date: "2026-05-19T14:00:00Z",
result: pass, runner-id: "test-runner@1.0", evidence-refs: ["<прогон>"] }
```

SR-01..04 «верифицированы» не сами по себе, а как часть пары (версия 1.0, сборка) — отдельного статуса у них нет. Версия 1.0 готова к предъявлению.

Соответствие **контракту** проверяет отдельный слой — **AT** (standard/09 §9.19): приёмочные тесты, которые изолированный агент выводит только из итогового T3, не заглядывая ни в ADAPT, ни в SR, ни в код. TC доказывают, что система соответствует интерпретации; AT — что интерпретация соответствует договору. Пока не все AT зелёные, продукт к сдаче не предъявляется (§10.4.3).

Цепочка трассировки

В любой момент восстанавливается происхождение артефакта:

```
TC-02 (negative)
  └─ verifies SR-01 (sign-up)
       └─ derived from ADAPT-001 §2 Forward (email/password)
            └─ interprets TZ-2026-001 §2 (immutable)
            └─ B-001 decided-in ACTZ-001 §1 (signed: клиент + исполнитель)
       └─ constrained-by SPEC-API-01 (REST contract)
                                └─ depends-on SPEC-DATA-01, SPEC-SEC-01
```

```
AT-01
  └─ verifies TZ-2026-001 §2 + ACTZ-001 §1   # только контракт, без внутренних
артефактов
```

Аудит через год: «откуда взялось требование возвращать 422 на невалидный email» — TC-02 → SR-01 → ADAPT-001 §2. Трассировка полная.

Что вы только что сделали

- Создали неизменяемый договорной артефакт (TЗ).
- Прошли двустороннюю интерпретацию через ADAPT с тремя backward findings → вынесли вопросы клиенту протоколом ACTZ → подписанные решения вернулись в ADAPT через `decided-in` → approved.
- Получили итоговое TЗ (начальное TЗ + подписанный ACTZ-001) — эталон будущей сдачи-приёмки.
- Декомпозировали в BR + 4 SR, привязанные к approved ADAPT.
- Описали 3 SPEC (API, DATA, SEC) как параллельную ось структуры.
- Покрыли каждый SR pos+neg TC парой (минимум 8 TC).
- Прошли три шлюза качества: QG-ADAPT-approve (≡ QG-3 Architecture), QG-0 комплекта (версия 1.0) и QG-2 Verification Gate (запись верификации версии 1.0).

Это полный цикл RENAR в минимальной форме. На реальном проекте — больше BR/SR/SPEC, больше backward findings, делегирование на AI-агентов; опционально QG-4 (acceptance).

В реальной работе шаги 2-7 выполняет AI-агент. Инженер не пишет frontmatter и body артефактов построчно — он формулирует задачу, читает результат, уточняет и утверждает. Это штатный режим (standard/00 §0.2.1). Полный сценарий первичного TЗ и delta-TЗ с adversarial reviewer — в 01-walkthrough.md фаза 2 + фаза 8.

Что дальше

Хотите...	Документ
Детальный сквозной пример (login + 2FA, фазы 0–9)	01-walkthrough.md
Переход с legacy подхода на RENAR	02-transition-guide.md
Git как носитель (commit-policy, PR-ревью, hooks)	03-tool-guide-git.md
Документо-ориентированный носитель	04-document-store-substrate.md
Сравнение RENAR с SAFe / BABOK / ISO 29148	05-safe-comparison.md
Compliance: GDPR / ФЗ-152 / AI Act	06-compliance.md
Failure modes — типовые провалы и паттерны	07-failure-modes.md
Полная нормативная спецификация (16 глав)	`standard/`
Схемы артефактов и validation rules	`reference/02-schemas.md`

Глоссарий и mapping на отраслевые стандарты	`reference/01-glossary.md`

01. Сквозной пример: Login Flow для AcmeCorp

> *Один полный цикл RENAR от подписанного ТЗ до accepted release. Пример — внутренний инструмент с регистрацией через корпоративный email и 2FA. Цель — показать **все этапы** на одном среднем по размеру проекте.*
>
> ***Контекст:** AcmeCorp, ~1 спринт работы команды, стек Next.js + FastAPI + PostgreSQL. RENAR-зрелость уровня RENAR-3+ (полный ADAPT + TC + adversarial). Пример **независим от вида хранилища**: операции через capabilities V1–V6; конкретная раскладка каталогов — 03-tool-guide-git или 04-document-store-substrate.*
>
> ***Предпосылки:** 00-quickstart, core/renar-core, reference/01-glossary.*

Фазы — иллюстрация, не норма. Порядок фаз 0–9 — один из возможных порядков работы; стандарт нормирует предусловия и правила, а не последовательность (standard/01 §1.2.1). Организация с иным процессом соответствует стандарту, пока предусловия каждого артефакта выполнены.

Маршрут читателя. Фазы 0–2 — сбор контекста, подписание ТЗ, ADAPT и протокол уточнения ТЗ (ACTZ). Фазы 3–4 — декомпозиция в BR/SR/SPEC и генерация пар pos/neg для TC. Фаза 5 — канонические шлюзы QG-0 (утверждение требований) и QG-1 (только переход TC `draft → ready`). Фазы 6–7 — реализация TR и верификация (QG-2). Фазы 8–9 — дельта-ТЗ при изменениях и приёмка: AT против итогового ТЗ плюс опциональный QG-4 по бизнес-результату.

Фаза 0 — Сбор требований (elicitation)

До подписания ТЗ. AI-агент проводит 2-3 интервью со stakeholder (Sales Director, IT Manager) и собирает контекст в структурированном виде.

Артефакты фазы 0 (справочные, не нормативные для RENAR Core): `elicitation/{domain-context.md, sales-director.yaml, it-manager.yaml, findings-clustered.md, critic-review.md, multi-model-diff.md}`. Фаза 0 не закреплена в Core — область методологии elicitation, вне scope RENAR v1.0 (standard/01 §1.3).

Фаза 1 — Импорт ТЗ

После итераций elicitation клиент подписывает `TZ-2026-042`:

```
# TZ-2026-042 — Login Flow для AcmeCorp Internal Tool
Дата подписания: 2026-05-03 · Стороны: AcmeCorp + VendorCorp

## §1. Цели
Сократить время входа сотрудников AcmeCorp в инструмент до <2 минут от первого
захода до полного доступа.
```

```
## §2. Функциональные требования
### ФТ-001. Регистрация по корпоративному email
Сотрудник регистрируется через email из домена @acmecorp.com. Email вне домена —
отказ с пояснением.

### ФТ-002. Двухфакторная аутентификация (TOTP)
После регистрации обязательная настройка 2FA через TOTP.

### ФТ-003. Восстановление доступа через корпоративного администратора
При потере 2FA устройства — recovery через ticket в IT support.

## §3. Нефункциональные требования
### НФТ-001. Производительность: Login <2 секунд (p95).
### НФТ-002. Безопасность: bcrypt cost-factor ≥ 12; логи входов — 1 год;
блокировка после 5 неудачных попыток за 15 минут.
### НФТ-003. Юрисдикция: все данные в РФ (гос-контракты).
```

ТЗ подписан → **неизменяемо**. Любые правки идут через ADAPT (фаза 2) или delta-TZ (фаза 8). Runtime **обязан** зарегистрировать неизменяемое ТЗ как ревизию (V1+V2) с AI-provenance (V6).

Фаза 2 — ADAPT и ACTZ (интерпретация и протокол уточнения)

Здесь расходятся два контура. **ADAPT** — внутренняя интерпретация: её аудитория инженер, агент и рецензент, подписывает её архитектор (standard/07 §7.5). **ACTZ** — протокол уточнения ТЗ: то, что вынесено клиенту и подписано обеими сторонами, то есть обязательство (§7.13). Граница проходит по аудитории: что показано клиенту и утверждено — обязательство; что не показано — интерпретация.

2.1 Primary agent генерирует draft ADAPT. Input: TZ-2026-042 (неизменяемо). Output: draft ADAPT-001 + Forward sections (по §2 ФТ + §3 НФТ) + Backward findings (6 candidates) + V6 provenance.

2.2 Adversarial review. Отдельный critic-agent (другая модель) проверяет backward findings; блокирует adapt-approve пока критические находки открыты. Примеры:

```
[HIGH] B-001 reclassify gap → hidden-assumption
[HIGH] missed backward: case-sensitivity email in ФТ-001
[MEDIUM] B-004 terminology: define "сотрудник" via User.role
[MEDIUM] B-006 feasibility: rate-limit scope (IP vs email vs session)
```

2.3 Iterative resolution. Архитектор корректирует Forward и Backward, AI re-генерирует. После 2 циклов adversarial: 7 backward записей (B-001..B-007), все reclassified либо подготовлены к выносу клиенту; Forward охватывает §2 + §3 ТЗ полностью.

2.4 ACTZ-001 — протокол уточнения ТЗ № 1. Пять находок из семи требуют решения клиента. Архитектор не пересылает сырой вывод агента: он агрегирует вопросы, переформулирует их на языке обязательств («домен сравнивается без учёта регистра»), прикладывает предложения — и выносит одним протоколом. Клиент подписывает; подписывает и исполнитель.

```
---
id: ACTZ-001
title: "Протокол уточнения ТЗ № 1 к TZ-2026-042"
type: ACTZ
tz-ref: TZ-2026-042
resolves: [B-001, B-002, B-004, B-006, B-007]
status: signed
client-signature: { signed-by: "Иванова А.А.", role: "Product Lead",
organization: "AcmeCorp", signed-at: "2026-05-04T11:30:00Z" }
vendor-signature: { signed-by: "Петров П.П.", role: architect, organization:
"VendorCorp", signed-at: "2026-05-04T11:50:00Z" }
---

## §1. Регистр email
Адрес `@AcmeCorp.com` и `@acmecorp.com` — один и тот же сотрудник: домен
сравнивается без учёта регистра.

## §2. Термин «сотрудник»
Сотрудник — учётная запись с ролью `employee` в корпоративном каталоге.

## §3. Граница блокировки
5 неудачных попыток за 15 минут считаются по паре «IP + email», не по сессии.
```

Каждая находка, требующая решения клиента, получает `decided-in: ACTZ-001 §M` — ссылку на пункт **подписанного** протокола. Находка не может стать `resolved` без такой ссылки, а ADAPT не может быть утверждён при незакрытых находках (§7.4.5).

2.5 ADAPT в статусе `approved`. Подпись — только архитектора: клиент этот документ не видел и не подписывал, его решения живут в ACTZ-001.

```
---
id: ADAPT-001
title: "Адаптация TZ-2026-042 — Login Flow AcmeCorp"
type: ADAPT
trigger-stage: import-tz
source-tz: { id: TZ-2026-042, signed-date: "2026-05-03", signed-by-client:
"AcmeCorp PM" }
status: approved
approval:
  # client-signature отсутствует по §7.5 — ADAPT не выносится клиенту
  architect-signature: { signed-by: "Петров П.П.", role: architect, signed-at:
"2026-05-04T12:00:00Z" }
generates-requirements: [BR-01, BR-02, SR-01, SR-02, SR-03, SR-04, SR-05, SR-06,
SR-07]
generates-specs: [SPEC-UI-01, SPEC-API-01, SPEC-DATA-01, SPEC-SEC-01]
open-questions-count: 0
resolved-questions-count: 7
ai-provenance: { generated-by: "anthropic-claude-opus-4-7@2026-05-04", prompt-
```

```
template: "prompts/adapt-from-tz.md@v2.1", human-edits: true }
---
```

После утверждения ADAPT-001 — **неизменяем**.

Итоговое ТЗ (§7.14) на этот момент: `TZ-2026-042` + `ACTZ-001`. Это и есть эталон, против которого система будет сдаваться в фазе 9. ADAPT в эталон приёмки не входит — клиент отвечает только за то, что подписывал.

Фаза 3 — Декомпозиция в BR / SR / SPEC

3.1 Декомпозиция. Operation: `decompose`. Input: approved ADAPT-001. Output: draft BR (2), SR (7), SPEC (4) + adversarial-review артефактов.

3.2 Adversarial-находки:

```
[HIGH] BR-01 stakeholder поле пустое — кто owner business goal?
[HIGH] SR-05 говорит "bcrypt cost-factor 12" — это deployment detail, должно быть
в SPEC-SEC-01, не в SR.
[MEDIUM] НФТ-003 (юрисдикция) не отражено в data-classification SR-01.
[MEDIUM] SPEC-UI-01 не имеет accessibility-level — WCAG-AA минимум для
корпоративного инструмента.
→ 4 находки → fix → re-generate.
```

3.3 Финальный набор артефактов:

```
acmecorp-requirements/
├── br/
│   ├── BR-01-self-service-registration.md
│   └── BR-02-secure-mfa.md
├── sr/
│   ├── SR-01-email-domain-validation.md       (ФТ-001)
│   ├── SR-02-totp-enrollment.md               (ФТ-002 setup)
│   ├── SR-03-totp-verification.md             (ФТ-002 verify)
│   ├── SR-04-password-recovery-via-admin.md   (ФТ-003)
│   ├── SR-05-rate-limiting-failed-logins.md   (НФТ-002)
│   ├── SR-06-audit-logging.md                 (НФТ-002 audit)
│   └── SR-07-data-residency-ru.md             (НФТ-003)
├── specs/
│   ├── ui/SPEC-UI-01-login-flow.md
│   ├── api/SPEC-API-01-auth.md
│   ├── data/SPEC-DATA-01-user-model.md
│   └── sec/SPEC-SEC-01-auth-policy.md
└── tz/TZ-2026-042.md
```

3.4 Пример: SR-01 (frontmatter + body):

```
---
id: SR-01
title: "Валидация домена email при регистрации"
type: SR
parent: { id: BR-01 }
source: { adapt: "ADAPT-001", adapt-section: "Forward §2.1", tz-section: "§2
ФТ-001" }
constrained-by: ["SPEC-API-01", "SPEC-DATA-01", "SPEC-SEC-01"]
data-classification: { contains-pii: true, data-residency: ["RU"], retention-
days: 365 }
compliance: [{ standard: "ФЗ-152", article: "ст.13.1" }]
ai-provenance: { generated-by: "anthropic-claude-opus-4-7@2026-05-04", prompt-
template: "prompts/decompose-adapt.md@v2.1", context-tokens: 12450, output-
tokens: 320, human-edits: true }
---

## Описание
Регистрация разрешена только если email принадлежит домену `@acmecorp.com`.
Остальные домены отклоняются с пояснением.

## Поведение
- email НЕ из `@acmecorp.com` → 422 с `{"error":"email-domain-not-allowed",
"allowed-domain":"acmecorp.com"}`. [ADAPT-001 Forward §2.1; TZ-2026-042 §2
ФТ-001]
- email из `@acmecorp.com` → стандартная регистрация (SPEC-API-01).
- Whitelist хранится в `SPEC-SEC-01.allowed-domains`, расширяется без релиза.
- Сравнение домена — case-insensitive (ADAPT-001 Forward §2.1).

## Ограничения
- `*.acmecorp.com` subdomain — отдельное решение архитектора (не входит).
```

3.5 SPEC-API-01 (фрагмент):

```
---
id: SPEC-API-01
title: "REST API аутентификации"
type: SPEC-API
source: { adapt: "ADAPT-001", adapt-section: "Forward §2" }
api-style: rest
api-version: "v1.0.0"
versioning-strategy: url-path
authentication: bearer-jwt
rate-limits: [{ endpoint: "POST /auth/login", limit: "5/15min/ip+email" }]
contract-file: { format: openapi-3.1, location: "contracts/auth-api.yaml" }
depends-on: ["SPEC-DATA-01", "SPEC-SEC-01"]
---

## Endpoints

### POST /auth/register
- body: `{"email": "<corp-email>", "password": "<strong>"}`
```

```
- 201 → `{"user_id": "<uuid>", "verified": false, "totp_setup": false}` · 422 →
invalid · 409 → email exists

### POST /auth/login
- body: `{"email", "password", "totp": "<6digits>"}`
- 200 → `{"access_token": "<jwt>", "expires_in": 3600}` · 401 → invalid · 429 →
rate limit

### POST /auth/totp-setup — см. SR-02.

## Error model
Единая структура: `{"error": "<code>", "details": {...}}`.
```

Фаза 4 — Генерация TC (пары pos/neg)

Operation: `tc-generate` SR-01 → pos/neg TC pairs per testable assertion (standard/09 §9.7: на каждое утверждение SR — минимум одна пара positive + negative TC).

TC-001 (позитивный):

```
---
id: TC-001
title: "Регистрация с разрешённым доменом — happy path"
type: TC
tc-type: system
verifies: [{ id: SR-01 }]
negative: false
automation: { status: automated, location:
"tests/auth/test_registration.py::test_allowed_domain_succeeds", runner: pytest }
---

## Контекст
SR-01, раздел «Требование»: регистрация с email из whitelist-домена создаёт
учётную запись.

## Предусловия
- БД пуста; email alice@acmecorp.com не зарегистрирован.

## Шаги
POST /auth/register {email: "alice@acmecorp.com", password: "ValidPass123!"}

## Pass-критерий
- status 201; body содержит {"user_id": "<uuid>", "verified": false,
"totp_setup": false}; User в БД создан; verification email отправлен (mock SES).

## Fail-критерий
- status ≠ 201; plaintext password в body/логах; User с другим email (case
mismatch); email верификации не отправлен.

## Постусловия
- User удалён seed-механизмом; очередь mock SES очищена.
```

```
## Out of scope
- TOTP setup → TC-005 (SR-02); rate limiting → TC-009 (SR-05).
```

TC-004 (негативный):

```
---
id: TC-004
title: "Регистрация с неразрешённым доменом — отказ с пояснением"
type: TC
tc-type: system
verifies: [{ id: SR-01 }]
negative: true
automation: { status: automated, location:
"tests/auth/test_registration.py::test_disallowed_domain_rejected", runner:
pytest }
---

## Контекст
SR-01, раздел «Поведение»: email вне whitelist-домена отклоняется с объяснимой
причиной.

## Предусловия
- email "bob@gmail.com" (вне whitelist).

## Шаги
POST /auth/register {email: "bob@gmail.com", password: "ValidPass123!"}

## Pass-критерий
- status 422; body == {"error": "email-domain-not-allowed", "allowed-domain":
"acmecorp.com"}; User в БД НЕ создан; email НЕ отправлен; audit-запись о rejected
attempt (для SR-06).

## Fail-критерий
- status ≠ 422; User создан; email отправлен (security leak); audit-запись
отсутствует.

## Постусловия
- Audit-запись удалена seed-механизмом.

## Out of scope
- Формат audit-записи → TC-021 (SR-06).
```

Фаза 5 — Шлюзы качества перед кодом (QG-0 и QG-1)

После генерации TC для всех 7 SR — суммарно 26 записей контрольных примеров (пары pos/neg + дополнительные негативы для SR-05, SR-06).

5.1 QG-0 — утверждение версии комплекта 1.0. BR-01, семь SR, SPEC и 26 TC-норм утверждаются одной версией (standard/10 §10.5.2). Предусловия: `source.adapt = ADAPT-001 (approved)`; дерево полно (у каждого SR — `parent`, у каждого SPEC — источник); adversarial-review успешен по каждому артефакту; на каждое утверждение — пара TC-норм; утверждения и связи cite разделы ADAPT-001. Постусловие: запись версии `1.0` с подписью Архитектора; отдельных статусов у BR/SR/SPEC нет.

5.2 QG-1 — допуск реализаций TC. По §10.3.2 QG-1 применим только к реализации TC и отделяет допущенный к прогону код проверки от ещё не готового. Предусловия: `automation.set-version: "1.0"` указывает на утверждённую версию (V5); `automation.status` + `location` валидны; статические проверки и сухой прогон пройдены; фиксирующий красный прогон записан; обязательные секции body заполнены. Постусловие: реализация допущена.

После **QG-0** версии 1.0 и **QG-1** на каждой реализации TC можно открывать работу по TR (фаза 6).

Фаза 6 — Реализация

> ***Порядок не случаен.*** *TC заморожены (`ready`) в фазе 5 — **до** первой строки кода, и писал их не тот агент, который сейчас пишет реализацию (standard/09 §9.18). Тест, рождённый рядом с кодом, проверяет код, а не требование. Отсюда же требование красной истории: перед реализацией TC прогоняется и обязан упасть — тест, ни разу не бывший красным, ничего не доказывает.*

6.1 Создание задач (TR). Operation `sync-tasks`: input — verified SR/SPEC set; output — 7 TR in implementation tracker (parent SR, implements-spec[], QG-0 ready с Goal + AC).

6.2 Разработчик берёт TR-101. QG-0 checks: Goal from SR-01; AC list (4 items); parent.id resolves (approved); implements-spec present; negative scenario in AC → work session allowed.

6.3 Реализация (фрагмент):

```
# acmecorp-login.src/src/auth/registration.py
from fastapi import HTTPException
from config import settings   # allowed-domains из SPEC-SEC-01

def validate_email_domain(email: str) -> None:
    domain = email.split("@", 1)[-1].lower()
    if domain not in settings.AUTH_ALLOWED_DOMAINS:
        raise HTTPException(status_code=422, detail={
            "error": "email-domain-not-allowed",
            "allowed-domain": settings.AUTH_ALLOWED_DOMAINS[0],
        })
```

```
# acmecorp-login.src/tests/auth/test_registration.py
def test_allowed_domain_succeeds(client, db, mock_ses):
    r = client.post("/auth/register", json={"email": "alice@acmecorp.com",
"password": "ValidPass123!"})
    assert r.status_code == 201
```

```
    assert "user_id" in r.json()
    user = db.query(User).filter_by(email="alice@acmecorp.com").one()
    assert user.verified is False
    mock_ses.send_email.assert_called_once_with(template_id="verification-email",
to_email="alice@acmecorp.com")

def test_disallowed_domain_rejected(client, db):
    r = client.post("/auth/register", json={"email": "bob@gmail.com", "password":
"ValidPass123!"})
    assert r.status_code == 422
    assert r.json() == {"error": "email-domain-not-allowed", "allowed-domain":
"acmecorp.com"}
    assert db.query(User).count() == 0
```

6.4 Хук валидации на стороне носителя:

```
[hook] Проверка связей TR-101: parent.id SR-01 (approved); implements-spec [SPEC-
API-01, SPEC-SEC-01].
[hook] Негативные TC: SR-01.verified-by включает TC-002, TC-004 (negative).
✓ Изменение разрешено.
```

Фаза 7 — QG-2 (шлюз верификации)

7.1 CI запускает TC. `pytest acmecorp-login.src/tests/auth/test_registration.py` → 4 TC PASSED → Bot обновляет `last-run.result = pass`, `last-run.set-version = 1.0` в TC файлах. В истории прогонов у каждого TC виден переход «красный → зелёный»: фиксирующий прогон до реализации был красным — это условие, при котором TC засчитывается как доказательство на QG-2 (§9.18.2).

7.2 Выборочная проверка (standard/09 §9.14). Раз в спринт инженер вручную запускает 5 passing TC, отобранных по машинным сигналам (приоритет — TC без красной истории, затем пережившие мутанта, затем класса `implementation-originated`; остаток — случайным добором), и сверяет фактический результат с SR. Selected: TC-001, TC-008, TC-012, TC-019, TC-024 → 5/5 совпадают.

7.3 Запись верификации версии 1.0. QG-2 предусловия: все TC версии прошли при `last-run.set-version = 1.0`; обе TC каждой пары; красная история; выборочная проверка пройдена. Постусловие: runner дописывает `verification` в запись версии 1.0 (§10.5.4) — SR-01 верифицирован как часть пары (версия 1.0, сборка), собственного статуса у него нет; обновляется индекс покрытия.

Фаза 8 — Дельта-ТЗ

8.1 Клиент через неделю:

```
# TZ-2026-051 — Дополнение к TZ-2026-042
Базовый: TZ-2026-042

## §2 (изменение) ФТ-001 (расширение)
Дополнительно разрешить @subsidiary.acmecorp.com (дочерняя компания). Whitelist
расширяется до 2 доменов.
```

8.2 Delta-ADAPT + ACTZ-002. Operation `adapt-from-tz (delta)` : input — TZ-2026-051 + parent ADAPT-001; output — draft ADAPT-001-delta-1 + delta Forward + backward findings (e.g. B-008 scope). Находка B-008 («считать ли `@sub.subsidiary.acmecorp.com` разрешённым?») выносится клиенту протоколом `ACTZ-002` — «Протокол уточнения ТЗ № 2»; после подписания B-008 получает `decided-in: ACTZ-002 §1` , и delta-ADAPT утверждается подписью архитектора. Итоговое ТЗ становится: `TZ-2026-042` + `TZ-2026-051` + `ACTZ-001` + `ACTZ-002` .

8.3 Анализ влияния. Operation `impact-analysis --delta TZ-2026-051` :

```
Affected:
  BR-01 (расширение охвата)
  SR-01: изменён в черновике → войдёт в версию 1.1
  TC-001..004: нормы изменены → реализации устареют (automation.set-version 1.0 ≠
1.1); +2 new TC (subsidiary domain)
  TR-115: new implementation task
  SPEC-SEC-01: allowed-domains extended
```

8.4 Apply delta. Архитектор открывает изменения с маркером `[delta:TZ-2026-051]` . AI обновляет SR-01 (расширяет whitelist), генерирует 2 новых TC. Реализация в TR-115. CI прогоняет TC, бот обновляет last-run. Черновик утверждается версией 1.1 (QG-0 комплекта); реализации пересоздаются против новой нормы (`automation.set-version: "1.1"`). После spot-check — запись верификации версии 1.1.

> ***Note (simple delta).*** *Если adversarial reviewer выносит вердикт «no findings, no clarifications» (§7.4.1.2),* ***delta-ADAPT не создаётся*** *— BR/SR/SPEC получают* `source.tz-section` *напрямую с зафиксированным* `adversarial-review-ref` *. Тривиальное изменение (переименование поля) не порождает ни интерпретации, ни протокола: спрашивать клиента не о чем.*

Фаза 8a — Три входа изменения комплекта

Дельта-ТЗ — не единственный путь, которым комплект описания получает новую версию. Изменение приходит тремя входами; различаются они моментом и тем, что уже известно, а не ценой: каждый даёт минорную версию сразу (standard/10 §10.5.1), задачи растут из изменённого комплекта, а не из текста входа.

Вход	Когда	Что известно	Провенанс в BR / SR / SPEC	Что производит
Дельта-ТЗ (фаза 8)	клиент изменил обещание	новый текст контракта	`source.tz-section` на дельту; при находках — ADAPT / ACTZ	версия комплекта; переделка с аннулированием верификации
Концепт на изменение	инженер видит доработку раньше, чем есть задача — до QG-0 задачи	целевое устройство: как должно быть	`source.adversarial-review-ref` + ссылка на принятый концепт в записи версии	версия комплекта (минор)
Обратная находка / проблема	в ходе задачи или после факта: провал ТС, дефект, замечание клиента	только «что не так»	зависит от исхода разбора (таблица ниже)	диагноз → дефект реализации либо версия комплекта

Концепт на изменение описывает целевое устройство — без сравнений с текущим, без вариантов и без открытых вопросов; он не входит в комплект и клиентом не подписывается. Принятый Архитектором концепт **вносится в комплект** и даёт минор; дальше он — история версии, не источник истины. Отложенная реализация — запись в очереди без собственной версии комплекта. Устное поручение или пожелание входом не является: его оформляют концептом. Для решений, которые проверяются поведением экрана, а не текстом, концепт дополняет или заменяет кликабельный прототип на условных данных — в комплект входит только то, что из него внесено в SPEC-UI.

Исходы разбора проблемы. Разбор устанавливает, кто неправ, и только затем появляется дефект. Пять исходов и их место в RENAR:

Исход	Кто неправ	В RENAR	Комплект	Контракт
К — код	реализация	дрейф источника истины (standard/04 §4.11, класс 4.11.3) → TR против текущей версии	не тронут	не тронут
Т — тест	реализация ТС (норма верна)	реализация ТС пересоздаётся против той же нормы (standard/09 §9.9); красная история сохраняется	не тронут	не тронут
Э — описание	комплект	обратная находка внутреннего контура (standard/06 §6.13.2): `gap`, `hidden-assumption`, `contradiction`, `terminology`, `feasibility` без наблюдаемых клиентом последствий	минор сразу либо запись в очереди	не тронут
И — обещание	контракт	обратная находка с контрактным исходом: `scope`, `contradiction`, `regulatory` или	версия после подписи ACTZ; аннулирование	изменён

нужно поменять		любая категория с наблюдаемым клиентом поведением → проект ACTZ, двусторонняя подпись (standard/07 §7.13)	верификации затронутых ТС	
Н — нужно то, чего не обещали	вне заказа	`scope` (не входит) → дельта-ТЗ либо новый ТЗ; в текущий комплект не попадает	не тронут	новый

Отсев — «требование вне объёма реализации» — проблемой не становится и записи не оставляет. Граница, которую агент не переходит: исходы **Э, И, Н объявляет только человек** — Архитектор для Э, уполномоченное лицо исполнителя вместе с Архитектором для И / Н (standard/05 §5.4); агент предлагает диагноз и готовит проект ACTZ, но версию не утверждает и подписи не ставит. К и Т агент разбирает и чинит сам: комплект не тронут, правка идёт обычной задачей.

Фаза 9 — Приёмка: AT против итогового ТЗ + QG-4

9.1 Перегенерация AT. Перед испытаниями изолированный агент заново выводит приёмочные тесты **AT** из действующей редакции итогового ТЗ (standard/09 §9.19). На вход ему подаётся только `TZ-2026-042` + `TZ-2026-051` + `ACTZ-001` + `ACTZ-002`. Доступа к ADAPT, BR/SR/SPEC, ТС и коду у него нет, и модель его отличается от модели основного агента — иначе он воспроизведёт ту же интерпретацию, и приёмка выродится в повторную верификацию.

```
---
id: AT-03
title: "Регистрация из домена дочерней компании"
type: AT
verifies: [{ tz: "TZ-2026-051 §2" }, { actz: "ACTZ-002 §1" }]
tz-version: "effective@2026-05-14"
generator: { vendor: "<vendor-B>", model: "<model-B>", internal-context-access:
false }
negative: true
status: passing
environment-ref: SPEC-TEST-01
automation: { kind: dynamic, location: "at/test_subsidiary_domain.py", runner:
pytest }
---

## tz_text
«Дополнительно разрешить @subsidiary.acmecorp.com (дочерняя компания). Whitelist
расширяется до 2 доменов.»
```

Дословная цитата пункта контракта рядом с шагами проверки — обязательная часть тела AT: на испытаниях предъявляется сам пункт договора, а не его пересказ.

9.2 Релизный гейт. Продукт не предъявляется к сдаче, пока не все AT в статусе `passing` (§10.4.3). Прогон даёт 11/12 зелёных: `AT-07` («восстановление доступа через администратора») красный при всех зелёных ТС. Это самая ценная строка таблицы маршрутизации (§9.19.5): AT ✗ / ТС ✓ — ошибка **интерпретации**, а не кода. Разбор подтверждает: ADAPT свёл recovery к сбросу пароля,

тогда как ТЗ требует ticket в IT support. Чинится ADAPT (errata) и производные SR/TC, после чего AT-07 зеленеет. Отчёт `ACCEPTANCE.md` — покрытие AT по разделам итогового ТЗ — актуализируется автоматически.

9.3 QG-4 (опционально) — бизнес-результат. Через 4 недели после релиза измеряется бизнес-эффект. Гейт не подменяет приёмку по контракту и с ней не смешивается: цель может быть достигнута при несоответствии договору, и наоборот.

```
BR-01 KPI: Time-to-first-login — target <2min P95, actual 1.4min (143%)
BR-02 KPI: 2FA adoption — target ≥95%, actual 97%
```

9.4 Sign-off. Клиент принимает результат: BR версии комплекта принят (QG-4), отметка — в записи версии. Архив: `ACCEPTANCE.md` + `QG-4-REPORT-v1.0.md` + lessons learned `lessons/2026-Q2.md`.

Финальные артефакты

```
acmecorp-requirements/
├── adapt/                ADAPT-001-main.md (frozen) + ADAPT-001-delta-1.md
(frozen)
├── actz/                 ACTZ-001.md + ACTZ-002.md (signed, immutable)
├── br/                   BR-01 + BR-02 (status: accepted)
├── sr/                   SR-01 (v1.1, verified) + SR-02..SR-07 (v1.0, verified)
├── specs/                SPEC-UI-01 + SPEC-API-01 + SPEC-DATA-01 + SPEC-SEC-01
(verified; SPEC-SEC-01 v1.1)
├── tests/                TC-001..TC-028 (28 TC, 100% passing)
├── at/                   AT-01..AT-12 (перегенерированы перед испытаниями, 100%
passing)
├── tz/                   TZ-2026-042.md + TZ-2026-051.md (delta, immutable)
├── elicitation/          # артефакты фазы 0
├── lessons/2026-Q2.md    # уроки фазы 9
├── ACCEPTANCE.md         # покрытие AT по итоговому ТЗ
└── QG-4-REPORT-v1.0.md   # отчёт по бизнес-результату
```

Итоговое ТЗ здесь — не файл, а вычисляемая сумма: `TZ-2026-042` + `TZ-2026-051` + `ACTZ-001` + `ACTZ-002`. Именно она подавалась на вход генератору AT.

Метрики проекта

Метрика	Значение
RDLT (TZ signed → all SR verified)	11 days
Coverage Velocity	100% за 2 спринта

Hallucination Rate (детектированных)	0%
Найдено adversarial-находок (цикл 1)	4 high + 2 medium
Test-spec drift на delta-T3	0%
Подписанных ACTZ	2 (протоколы уточнения Т3 № 1 и № 2)
AT на испытаниях	12 (1 красный → ошибка интерпретации, исправлена)
Споры на приёмке	0 (1 находка, закрыта до подписания)
Cost per BR	$0.46 (gen) + $0.18 (critic) = $0.64
Total AI cost	~$8.50
BRs accepted	2/2
Дней до accept	35

Что показывает этот пример

1. **Прозрачность** — каждый артефакт имеет provenance, каждый переход — шлюз с явными условиями.
2. **Скорость** — декомпозиция approved ADAPT — десятки секунд + 2 цикла adversarial.
3. **Трассировка** — от строки в Т3 до passing TC за несколько операций запроса к носителю.
4. **Дельта-Т3** — затронутые SR/SPEC/TC/TR вычисляются автоматически.
5. **Два контура вместо одного** — клиент подписывает ACTZ (обязательства), архитектор подписывает ADAPT (интерпретация). Спор «уточнение это или уже изменение» не возникает: достаточно спросить, выносилось ли решение клиенту.
6. **Приёмка против контракта** — AT, выведенные изолированным агентом из итогового Т3, поймали ошибку интерпретации, которую все 28 зелёных TC пропустили в принципе.
7. **Замыкание контура** — QG-4 связывает результат с бизнес-метриками (KPI achievement).
8. **AI-нативность** — критик и генератор — разные модели (изоляция); выборочная проверка находит расхождения, которые может пропустить только автоматический прогон.

Что дальше

- 02-transition-guide.md — переход с legacy подхода.
- 03-tool-guide-git.md — git как носитель.
- 04-document-store-substrate.md — документо-ориентированный носитель.
- 05-safe-comparison.md — сравнение с SAFe / BABOK / ISO 29148.
- 06-compliance.md — compliance mapping (GDPR / ФЗ-152 / AI Act).
- 07-failure-modes.md — failure modes.

Сквозной пример RENAR 1.1 — renar.tech

02. Переход на RENAR

> *Команды редко начинают с чистого листа. Эта глава — про то, как перевести существующий проект с ручного цикла «ТЗ → код» на RENAR постепенно, шаг за шагом, без остановки разработки. Каждый уровень даёт осязаемую пользу; команда выбирает, какого из `RENAR-N` достигать исходя из реальной ценности, а не из гонки за «полным соответствием» объявлениям на бумаге.*
>
> ***Предпосылки:*** *RENAR Core, standard/11-maturity-model (закрытый список уровней RENAR-1..RENAR-5), guide/07-failure-modes. Уже на раннем RENAR с legacy типами (`INT-TC`, `AIC`, ...) — см. 10-migration-v1.*

1. Оценка: где команда сейчас

Прежде чем мигрировать, оцените текущее состояние. Чек-лист «pre-RENAR»:

Признак	Pre-RENAR	RENAR-1 минимум
ТЗ существует как артефакт	В чате / Google Doc / Notion	Зафиксировано в носителе как файл
Кто-то ведёт требования	Только в трекере (Jira / Linear)	В носителе как BR / SR / ADAPT
Откуда взялось требование	По памяти / переспрашиваем	Любой может найти источник в носителе
Изменения требований	Устно на дейли	Через явный change-set (delta-ТЗ)
Тесты	В коде, привязки к требованиям нет	ТС как артефакты или хотя бы в коде с упоминанием SR-ID

Если 3+ строк в колонке «Pre-RENAR» — команда **до RENAR-1**. Это нормальная стартовая точка для большинства проектов.

1.1 Сигналы готовности

Команда готова к миграции, если:

- Болит, что требования теряются между чатами / тикетами / документами.
- Disputed acceptances случаются регулярно — «мы не это просили».
- При onboarding нового инженера месяцы уходят на восстановление контекста.
- Используется AI для генерации требований / кода, но нет систематической проверки.

Если ничего из этого не болит — RENAR может быть преждевременной оптимизацией. См. §8 «когда не нужен».

2. Этап 1 — войти в RENAR-1 (Стихийный, Ad-hoc)

Цель уровня: требования живут в носителе, не в чате; каждое ТЗ прошло состязательный обзор с выпущенной AR, а ADAPT создан при вердикте «findings present».

Типичная длительность: 1-2 недели.

2.1 Что добавляется

1. Выбирается и заводится **носитель** артефактов (`substrate`): каталог `<project>.req/` в репозитории, рабочая область document store или иной носитель — RENAR к конкретной реализации не привязан; см. возможности **V1–V6** (глоссарий §2.7).
2. Существующее ТЗ переносится в эту среду как не подлежащий произвольным правкам артефакт (с датой и подписями).
3. Каждое ТЗ проходит состязательный обзор; его исход выпускается как `AR` — запись состязательного обзора (standard/07 §7.4.6). При вердикте «findings present» создаётся `ADAPT` — артефакт-мост: раздел прямой интерпретации («как мы поняли») и раздел обратных находок. При вердикте «no findings» ADAPT не создаётся, а BR / SR / SPEC ссылаются на ТЗ напрямую через `source.tz-section` + `source.adversarial-review-ref` (§7.4.1).
4. Новые требования заносятся как BR / SR файлы в носителе, не только в трекер.

2.2 Что НЕ требуется на этом этапе

- Стандартизированный frontmatter (минимум — `id` + `title`).
- Lifecycle статусы (`draft` / `approved` /...) — артефакты могут жить без явных переходов.
- TC как отдельные артефакты.
- Hooks носителя.
- Нативный для носителя COVERAGE.

2.3 Типичные блокеры

- **«У нас уже есть тикеты в Jira»** — оставьте. RENAR-1 не требует миграции; tracker и носитель могут сосуществовать. Главное — носитель теперь источник истины для **новых** требований.
- **«ADAPT — лишняя работа»** — на одно ТЗ ADAPT занимает 1-2 часа. Это окупается на первой же спорной приёмке.
- **«Где хранить?»** — любой носитель, удовлетворяющий **V1–V6**. См. guide/03-tool-guide-git или guide/04-document-store-substrate.

2.4 Когда переходить на RENAR-2

Когда **новые требования** перестали уходить в чаты и начали стабильно появляться в носителе. Это занимает 4-6 спринтов привычки.

---

3. Этап 2 — перейти на RENAR-2 (Зафиксированный, Documented)

Цель уровня: структура и frontmatter; delta-ТЗ как явный change-set.

Типичная длительность: 2-4 недели после RENAR-1.

3.1 Что добавляется

1. Каждый BR / SR получает frontmatter с обязательными полями (см. reference/02-schemas).
2. Папки структурируются: `br/`, `sr/`, `adapt/`, `actz/`, `dpia/`, ...
3. Изменения требований — только через delta-ТЗ (новый неизменяемый артефакт), не через прямое редактирование.
4. ТЗ зафиксировано как неизменяемое (изменения только через delta-ТЗ).
5. Уточнения, полученные от клиента без нового ТЗ, оформляются **ACTZ** — протоколом уточнения ТЗ с двусторонней подписью (standard/07 §7.13), а не письмом в чате.

Что на этом шаге меняется по сути. До RENAR у команды один договорной документ — ТЗ, и всё, что обсуждалось после его подписания, живёт в переписке. RENAR разделяет две вещи, которые раньше сливались:

Что	Куда	Кто подписывает
Решение, вынесенное клиенту и им принятое	ACTZ («Протокол уточнения ТЗ № N»)	Клиент **и** исполнитель
Толкование ТЗ инженером — термины, достроенные сценарии, находки	ADAPT	Только архитектор (§7.5)

Клиент не подписывает ADAPT: раньше это была фикция — подпись под инженерным документом, который клиент не читал. Практический выход — **итоговое ТЗ** (§7.14): начальное ТЗ плюс все подписанные ACTZ. Именно оно, а не переписка и не ADAPT, становится эталоном сдачи-приёмки.

3.2 Шаблон: легализация существующих требований

Старые требования (которые уже были в носителе с RENAR-1) — пройти рецензирование и проставить frontmatter «как есть» без пересмотра содержания. Это **legalization**, не **rewrite**:

```yaml
---
id: BR-12
title: "Регистрация сотрудника через корпоративный email"
# статуса нет — BR входит в утверждённую версию комплекта (уже работает в проде)
created-at: "2025-12-01"  # ретроактивно
priority: must            # ретроактивная оценка
legacy: true              # маркер: требование не проходило ADAPT с нуля
---
```

Поле `legacy: true` опционально, но рекомендовано — отличает «исторические» требования от новых, которые с самого начала прошли весь RENAR-пайплайн.

3.3 Типичные блокеры

- **«frontmatter на 100 требований — это месяц»** — да. Делайте в фоне, по 10-15 требований / неделю; новые требования сразу с frontmatter.

- **«Delta-T3 замедляет»** — на первых неделях да. После 2-3 итераций обычно становится быстрее, чем «давай просто поменяем».
- **«Не понимаем, какой priority ставить ретроактивно»** — оставьте `priority: should` для всего legacy; явно `must` ставится только при подтверждении со stakeholder.

3.4 Когда переходить на RENAR-3

Когда frontmatter валиден для 80%+ артефактов и команда привыкла к delta-T3.

---

4. Этап 3 — перейти на RENAR-3 (Учитываемый, Tracked)

Цель уровня: автоматическая валидация + lifecycle enforcement + TC покрытие для priority=must.

Типичная длительность: 4-8 недель после RENAR-2.

4.1 Что добавляется

1. Hooks носителя валидируют frontmatter по schema на каждое изменение. Невалидные — блок integration.
2. Жизненный цикл ведётся реально: версии комплекта описания выпускаются (`draft` → `approved N.M` → `superseded`), у задач и ADAPT — собственные закрытые состояния (standard/10 §10.5).
3. TC создаются для всех priority=must BR / SR / SPEC.
4. Нативный для носителя COVERAGE report auto-generated на каждый выпуск версии комплекта и каждую запись верификации.
5. Reference-validation hook: создание BR / SR с ссылкой на ADAPT в статусе ниже `approved` — блок.
6. Реализация ссылается на BR / SR / SPEC с pinned `verifies[].version` .

4.2 Шаблон: backfill TC

Для всех priority=must требований без TC:

1. Отсортировать по «частоте упоминания в incident reports» — где TC даст максимум value.
2. Покрывать по 3-5 требований / спринт, не пытаясь backfill сразу всё.
3. `TC` создаются как артефакты в вашем носителе со связью `verified-by` .

4.3 Типичные блокеры

- **«Старые требования упрямо не приводятся к schema»** — оставьте их в legacy-папке; новые сразу schema-valid. Hook носителя применяется только к новым / изменяемым артефактам.
- **«Hooks замедляют PR cycle»** — измерьте: если > 30s — оптимизируйте hooks (parallelization, caching); не «отключить».
- **«Команда обходит hooks через `--no-verify` »** — это organizational failure pattern; root cause — hooks слишком медленные / шумные. Чините, не запрещайте.

4.4 Когда переходить на RENAR-4

Когда COVERAGE для priority=must = 100%, frontmatter валиден везде, lifecycle действительно работает.

---

5. Этап 4 — перейти на RENAR-4 (Верифицируемый, Verified)

Цель уровня: для всех утверждений утверждённой версии комплекта есть успешно пройденные контрольные примеры (`TC`); запись верификации версии под контрольной точкой `QG-2` (`Verification Gate`) обеспечивается носителем; соблюдена парность позитивных / негативных проверок; для материала от ИИ задан блок `ai-provenance` .

Типичная длительность: 6-12 недель после RENAR-3.

5.1 Что добавляется

1. 100% BR / SR / SPEC утверждённой версии комплекта имеют `verified-by` ссылку на ≥ 1 TC-норму той же версии.
2. Pos/neg парность для каждого нормативного утверждения.
3. Контрольная точка `QG-2` (`Verification Gate`) блокируется встроенными в носитель проверками: запись верификации версии — только при всех успешных `TC` с `last-run.set-version` , равной версии.
4. Все TC автоматизированы или явно `manual-pending` с deadline.
5. Для `tc-type: ux` — VLM-judge isolation.
6. Для `tc-type: eval` — judge-модель ≠ модель реализации.
7. `ai-provenance` для AI-сгенерированных артефактов — в составе канонического `standard/04 §4.10.1` : `generated-by` (с версией модели), `generated-at` , `prompt-template` , `context-tokens` , `output-tokens` , `human-edits` .
8. Source citation — каждое нормативное утверждение имеет pointer на источник в T3 или ADAPT.
9. Привязка согласования (reconciliation) запускается носителем не реже одного раза в неделю.
10. Spot-check 5 случайных passing TC раз в спринт.
11. Изоляция авторства теста: TC-норма утверждена в версии и реализация допущена (QG-1) до старта реализации поведения, и пишет его не тот агент, который пишет код (standard/09 §9.18).
12. Красная история: фиксирующий прогон TC до реализации обязан быть красным. Зелёный тест до кода — сигнал разбора, а не успех.

Приёмка против контракта. На этом же этапе вводится второй слой проверки — **AT** (standard/09 §9.19). TC доказывают соответствие интерпретации; AT — соответствие итоговому T3. Их генерирует изолированный агент, которому на вход подаётся **только** итоговое T3 (без ADAPT, SR, TC и кода) и модель которого отличается от модели основного агента. AT перегенерируются перед каждыми испытаниями; пока не все AT зелёные, продукт к сдаче не предъявляется (§10.4.3).

Практический смысл: команда с зелёными TC и красным AT видит ошибку **толкования** T3 — тот класс дефектов, который никаким TC не ловится, потому что TC выведены из той же интерпретации, что и код.

5.2 Шаблон: поэтапное покрытие

Достижение 100% verified-by покрытия — не одним PR. Подход:

1. Сначала pos-only TC для всех priority=must (бенефит — быстрая обратная связь).
2. Затем neg-TC pairs (бенефит — отлов test-fitting drift).
3. Затем `tc-type` extensions (ux / eval / contract / security) по мере необходимости.

5.3 Типичные блокеры

- **«Изоляция judge-модели дорого»** — это правда. Но это структурный rate limit на AIR-06 test-fitting drift — нельзя обойти без потери verification integrity.
- **«Source citation замедляет авторов»** — автоматизируйте: AI-генератор должен сразу выдавать citation; ручная авторизация — только для legacy backfill.
- **«Находки reconciliation заваливают команду»** — tunable thresholds; начинайте с conservative defaults, постепенно ослабляйте.

5.4 Когда переходить на RENAR-5

Когда RENAR-4 стабильно работает 2-3 квартала, метрики стабильны, команда не «горит» от reconciliation noise.

---

6. Этап 5 — перейти на RENAR-5 (Оптимизирующий, Optimized)

Цель уровня: adversarial review как gate; multi-model agreement для priority=must; cost/latency budgets; knowledge graph как primary search; continuous evaluation для AI-критических компонентов.

Типичная длительность: 12+ недель после стабильного RENAR-4.

6.1 Что добавляется

1. Adversarial critic — обязательный gate для `draft → approved`. Critic — модель, отличная от модели генерации.
2. Multi-model agreement для priority=must: артефакт генерируется ≥ 2 моделями; расхождения помечены и обязательны к разбору.
3. Cost / latency budget per artifact; превышение → автоматическая декомпозиция.
4. Knowledge graph как primary search для AI-агентов.
5. Continuous evaluation для SPEC-AI.
6. Метрика Hallucination Rate < 1%.
7. Метрика Multi-model Disagreement Rate отслеживается.
8. Возврат улучшений шаблонов в `requirements-library` — стандартная практика.

6.2 Когда RENAR-5 нужен

- Регулируемые отрасли (финтех, healthcare, AI-системы high-risk per AI Act).
- Когда продукт критически зависит от качества AI-генерации (генеративные продукты).
- Когда есть бюджет на continuous evaluation infrastructure.

Многие проекты могут остановиться на RENAR-4 — этого достаточно для **соответствия заявленному профилю** RENAR (conformance). RENAR-5 не обязательно «лучше», зато почти всегда **дороже и строже**. Выбирайте по потребности, а не по моде.

---

7. Миграция legacy-требований

Существующие тысячи требований в Jira / Confluence — как втягивать?

7.1 Стратегия не-миграции

Если legacy требования не активно меняются — **не мигрируйте**. RENAR применяется только к новым требованиям и активно изменяемым. Legacy остаётся в исходном месте как read-only «исторический контекст».

7.2 Стратегия выборочной миграции

Для legacy, которые активно меняются:

1. На момент первого change — затянуть в носитель как BR/SR с `legacy: true` маркером и минимальным frontmatter.
2. Применить change как delta-T3.
3. Через 1-2 итерации требование «созревает» — становится full-RENAR (без legacy маркера, с полным frontmatter и TC).

7.3 Стратегия bulk-import

Когда нужно мигрировать сразу > 100 требований (например, при organizational reorganization):

1. Скриптовая выгрузка из tracker → frontmatter скелет (id, title, минимум).
2. Все импортированные требования — `legacy: true` + `status: approved` (как есть в проде).
3. Backfill TC и full frontmatter — спринт за спринтом, не блокируя current work.

---

8. Когда RENAR НЕ нужен

RENAR — overhead. Не применяйте к:

- **One-off скриптам и прототипам** — не доходят до production, или становятся production только через rewrite.
- **Очень маленьким командам (1-2 человека)** — overhead управления носителем может превышать benefit.
- **Проектам без AI-генерации требований / тестов** — главная ценность RENAR (защита от AI-specific failure modes) теряется.
- **Краткосрочным экспериментам (≤ 1 спринт)** — не успеете окупить ADAPT-setup.

Минимальный порог окупаемости: проект с ≥ 3 итерациями требований, ≥ 2 разработчиков, использует AI где-либо в pipeline (генерация требований / кода / тестов / документации).

---

9. Антипаттерны миграции

Чего избегать:

9.1 Миграция «big-bang»

Симптом: Команда останавливает разработку на 2 спринта чтобы «перейти на RENAR». Через 2 спринта получают burnout и брошенную миграцию.

Вместо: Гибридный режим. Новые требования сразу в RENAR, старые — по мере касания. Долго, но устойчиво.

9.2 Пропуск уровней

Симптом: «Давайте сразу на RENAR-4». Команда пропускает RENAR-2/3, ставит full hooks + ai-provenance + adversarial critic, но frontmatter не валиден и lifecycle хаотичен.

Вместо: Уровни идут по порядку. RENAR-4 без RENAR-3 базы — не работает.

9.3 Паралич «идеального frontmatter»

Симптом: Команда тратит недели на полирование одного frontmatter — «а правильно ли я выбрал `priority`?» Реальная работа стоит.

Вместо: frontmatter — «достаточно хорошо». Ошибки правятся в delta-Т3. Главное — что-то быть в носителе, а не вылизанность.

9.4 Частичное принятие носителя

Симптом: Половина требований в носителе, половина — всё ещё в Jira. Никто не знает, где источник истины.

Вместо: Single source of truth должен быть **сразу**. Если нельзя перенести всё — перенесите только новые, явно объявите носитель владельцем «всего нового».

9.5 Подход «сначала tooling»

Симптом: Команда полгода пишет внутренние tooling вокруг RENAR (свой validator / своя CI / своя UI), не используя сам стандарт.

Вместо: Сначала практика на минимальном носителе (даже плоская папка с markdown). Tooling — когда становится больно вручную.

Стоимость и выгода внедрения (иллюстративно)

> *Секция **иллюстративная и ненормативная** — помогает прикинуть порядок величин. Конкретные числа зависят от проекта; реальную модель калибруйте по своим данным.*

Стоимость внедрения (что вкладывается):

- Обучение команды по RENAR Core — порядка полдня-дня на инженера.
- Состязательный обзор каждого Т3 и дисциплина ADAPT там, где обзор дал находки, — дополнительные часы на обратные находки до старта кода (окупаются отсутствием переоткрытия Т3 позднее).
- Носитель уже есть (git) — отдельных лицензий не требуется; tooling-автоматизация опциональна и вводится постепенно.

Выгода (что возвращается):

- Сокращение времени декомпозиции ТЗ с AI-ускорением (порядок величин — standard/12 §12.5.1: 5–10× на RENAR-4, иллюстративно).
- Меньше споров на приёмке: подписанный ACTZ фиксирует обязательство до кода, а сдача идёт против итогового ТЗ, а не против чьей-то памяти о переписке.
- Меньше инцидентов из негативных сценариев — за счёт обязательной pos/neg-парности ТС.
- Журнал аудита «что сдавали по контракту» — бесплатно из носителя (V1 / V6).

Когда не окупается: короткоживущие прототипы, чистый продуктовый дискавери без договора, команды без compliance-давления и без внешнего клиента (см. §8 «когда RENAR не нужен»). Для них достаточно RENAR Core или ничего.

> *Количественная ROI-модель (порядок экономии на проект и т.п.) пока живёт в research-материалах; перенос откалиброванной версии в этот раздел запланирован в **бэклоге фазы 8** для v1.1.*

---

10. Связанные документы

- standard/11-maturity-model — нормативные определения RENAR-1..RENAR-5 уровней (closed list).
- 00-quickstart — 30-минутный sample для маленького проекта.
- 01-walkthrough — полный example на одном проекте.
- 07-failure-modes — что может пойти не так при миграции; organizational failure patterns §5.
- reference/02-schemas — frontmatter schemas, обязательные для RENAR-2+.
- 03-tool-guide-git — специфичный для носителя guide для git.
- 04-document-store-substrate — informative обзор носителя document-oriented store.

---

11. Зафиксированные решения для v1.0

- **Legacy backfill bulk-import — bypass запрещён.** На initial import артефакты вносятся со статусом `imported-legacy` (специфичный для носителя marker, не входит в нормативный closed list lifecycle §10.5–§10.8) и не участвуют в conformance assessment до промотирования через нормальный QG-0 flow. Это сохраняет §1.7.3 declared-weaker запрет: validation не обходится, а лишь отложена до момента, когда артефакт начнёт claim conformance.
- **Откат с уровня RENAR-3 → RENAR-2 допустим** через formal downgrade per §13.8.2: выпускается новая версия manifest с пониженным `level`. План восстановления **не** обязателен (downgrade — намеренный, не потеря соответствия), но рекомендуется зафиксировать причину downgrade в журнале аудита.
- **Cross-org migration: каждая подсистема — свой manifest с собственным `level`** (§13.4). Organization-aggregate "RENAR-N" неформально определяется минимумом уровней подсистем; организационный manifest **не** нормируется RENAR v1.0.

11.1 Отложено на v1.1 (бэклог фазы 8)

- **Шаблоны миграции для команд 50+ инженеров.** Гайд ориентирован на группы 5–15 человек; для большего масштаба нужны полевые наблюдения. Ответственные: сопровождение стандарта RENAR и ранние внедренцы.

---

03. Носитель: VCS — git

> *Конкретная реализация RENAR на git. Структура* `.req` *репозитория, submodule-pinning между* `.req` *и* `.src`*, PR/MR ревью workflow, pre-commit hooks для capability V1-V6, delta-T3 workflow. Этот guide — informative; нормативное содержание (capability требования, schemas, lifecycle) — в* `standard/`*. Альтернатива на document-oriented store — guide/04-document-store-substrate.*
>
> ***Предпосылки:*** *standard/03-substrate-versioning (нормативные capability V1-V6), reference/02-schemas (frontmatter schemas).*

1. Когда выбрать git как носитель

Git — носитель по умолчанию для:

- Открытых стандартов и проектов (нет зависимости от внутренней инфры).
- Внешних клиентов без выделенной document-store инфраструктуры.
- Команд с устоявшимся PR-workflow.
- Проектов, где требования и код в одной экосистеме (один и тот же VCS provider).

Git **не** оптимален, если:

- Нужны частые конкурентные правки одного артефакта несколькими авторами без merge conflicts.
- Нужен built-in полнотекстовый поиск без внешних инструментов.
- Требуется UI для non-technical stakeholder (PM, юристов) без git CLI.

В таких случаях рассмотрите document-oriented store — guide/04.

2. Layout двух репозиториев

Каноническая структура — два связанных репо.

```
<project>/
├── <project>.req/                ← репозиторий ТРЕБОВАНИЙ (отдельный VCS repo)
│   ├── tz/                       ← TZ-YYYY-NNN.md (immutable после регистрации)
│   ├── actz/                     ← ACTZ-NNN.md (протоколы уточнения ТЗ,
двусторонняя подпись)
│   ├── adapt/                    ← ADAPT-NN.md (bridge artefacts)
│   ├── ar/                       ← AR-NNN.md (записи состязательного обзора,
§7.4.6)
│   ├── br/                       ← BR-NN.md
│   ├── sr/                       ← SR-NN.md
│   ├── specs/                    ← SPEC-* по подпапкам (arch/, api/, data/,
int/, …)
│   │   ├── arch/  api/  data/  ui/  ai/  int/  proc/  sec/  ops/  test/  doc/
│   ├── tr/                       ← TR-NN.md (task requirements)
```

```
│   ├── tests/                    ← TC-NN.md (контрольные примеры, `TC` —
самостоятельные артефакты)
│   ├── at/                       ← AT-NN.md (приёмочные тесты от итогового ТЗ)
│   ├── dpia/                     ← DPIA-NN.md (опционально для regulated)
│   ├── library/                  ← templates, patterns
│   ├── docs/                     ← AI-generated документация
│   ├── COVERAGE.md               ← auto-generated (`[coverage]` commits)
│   ├── REQUIREMENTS.md           ← auto-generated index
│   └── TEST-PLAN.md              ← auto-generated
└── <project>.src/                ← репозиторий РЕАЛИЗАЦИИ
    ├── src/                      ← код
    ├── tests/                    ← реализации TC (адресуются
`automation.location`)
    ├── requirements/             ← submodule → <project>.req @ <commit>
    ├── .gitmodules
    └── README.md
```

В `<project>.req/.gitattributes` для bot-generated артефактов:

```
COVERAGE.md      linguist-generated=true
REQUIREMENTS.md  linguist-generated=true
TEST-PLAN.md     linguist-generated=true
docs/**          linguist-generated=true
```

Это исключает их из статистики кода и сильно сжимает PR diff.

3. Capability mapping V1-V6 на git

Возможность (standard/03 §3.3)	**Норматив**	**Git-механизм**
V1 — неизменяемая история	Прошлые состояния артефактов нельзя переписать задним числом	Неизменяемая история коммитов; protected branch + запрет force-push на `main`; `id:` во frontmatter стабилен
V2 — атомарная единица изменения	Изменение применяется целиком либо не применяется	Атомарный commit / squash-merge PR как одна единица; delta-ADAPT = один PR
V3 — сравнение различий и рецензирование	Предложенное изменение рецензируется до утверждения	`git diff` + PR/MR review с обязательным approve до merge
V4 — ветвление / набор изменений	Черновик отделён от утверждённой правды	Ветка черновика комплекта против `main` (утверждённая версия); PR — набор изменений, слияние = выпуск версии `N.M`

V5 — сквозная фиксация версии	Реализация ссылается на конкретную версию комплекта	Submodule-pin между `.req` и `.src`; тег версии комплекта `renar-vN.M` = `set-version`; `automation.set-version` в TC
V6 — автор и отметка времени	Каждая единица изменения регистрирует автора и время	Метаданные коммита: автор + дата; для AI-агента — `ai-provenance`

> *RENAR-проверки на git (валидация схемы, контроль переходов статусов, coverage-отчёты, целостность ссылок, reconciliation / drift detection) — это **enforcement-механизмы** поверх возможностей, а не сами V1–V6. Их раскладка по pre-commit / CI — §3.1 и §8 этого гайда; нормативные классы дрейфа — standard/04 §4.11.*

> ***Состояние сценариев.** Приведённые ниже механизмы под `git` — **целевой образец v1.0**. Фактически готов только `scripts/validate-frontmatter.js`; остальные (`validate-lifecycle`, `validate-references`, `generate-coverage`, `detect-drift` и др.) — в **бэклоге фазы 8** (раздел §8 этого гайда). Пока скриптов нет, перечисленные возможности обеспечиваются вручную и код-ревью; автоматическое навязывание уровней RENAR-3+ «из коробки» под `git` пока недостижимо. Замечание относится и к document-store (guide/04).*

---

4. Submodule pinning

`<project>.src` фиксирует **конкретный commit** `<project>.req` через git submodule.

4.1 Как работает

- В `<project>.src` директория `requirements/` — submodule на `<project>.req`.
- При сборке / CI код знает: «я реализую требования по состоянию на commit `abc1234`».
- Разработчик в задаче открывает `requirements/sr/SR-05.md` через обычный `cat` или IDE — это файл в worktree.

4.2 Bump pattern

При обновлении требований:

1. PR в `<project>.req` с изменениями требований → ревью → merge.
2. **Отдельный** PR в `<project>.src`, который **только** двигает submodule pointer:

```
cd requirements
git pull origin main
cd ..
git add requirements
git commit -m "bump requirements: TZ-2026-042 delta + 3 new SR"
```

3. Этот PR явно показывает: «требования обновились до коммита X».

4.3 Почему submodule, не subtree / monorepo

- **Provenance:** для любого коммита кода точно известно, какую версию требований он реализовывал.
- **Изоляция ревью:** ревью требований (в `.req`) и ревью кода (в `.src`) не смешиваются.
- **Атомарность delta-T3:** delta — это атомарный PR в `.req` + последующий submodule-bump в `.src`. Нет «частично применённой delta».
- **Совместимость с document store:** при переходе на document-oriented store submodule pinning превращается в revision-token pinning — концепция та же.

Альтернативы (subtree, monorepo): рассматривайте только если submodule не работает по причинам, специфичным для вашего VCS provider; во всех остальных случаях submodule — рекомендация.

---

5. PR/MR review workflow

Два уровня ревью, разделённые по репозиториям.

5.1 Ревью в `<project>.req`

Фокус рецензента:

- Frontmatter schema-valid.
- Citation на T3 / ADAPT присутствует и валиден.
- `parent` / `verified-by` / `constrained-by` ссылки существуют.
- Lifecycle transition легитимна.
- Source citation в body для каждого нормативного утверждения (на RENAR-4+).
- Adversarial AI-review prompt пройден (на RENAR-5).

Утверждение = QG-0 (standard/10 §10.3.1).

5.2 Ревью в `<project>.src`

Фокус рецензента:

- Submodule pointer соответствует merged commit в `.req`.
- Реализация ссылается на утверждённую версию комплекта через `automation.set-version` (тег `renar-vN.M`).
- Новые / изменённые TC соответствуют контракту в `.req`.
- Никаких изменений Pass/Fail-критериев TC без `[test-spec-change]` тега.

Слияние ветки черновика в `main` = QG-0 комплекта (выпуск версии `N.M`, тег `renar-vN.M`); **QG-2** (standard/10 §10.3.3) — запись верификации версии после прохождения всех TC на ней.

5.3 Запрещённые анти-паттерны

- **PR одновременно в `.req` и `.src`** — нарушает изоляцию ревью; запрещён hook носителя.

- **Изменение submodule pointer без merged commit в `.req`** — pointer указывает в untracked commit; CI блокирует.
- **`[test-spec-change]` без отдельного approval** — Pass/Fail-критерий TC изменён вместе с code-fix; CI блокирует merge.

---

6. Pre-commit и pre-merge hooks

Минимальный набор hooks носителя для RENAR-3+ на git.

6.1 Pre-commit (в `<project>.req`)

```
# Запускается на коммит в .req
# Capability V2: schema validation
yamllint --strict $(git diff --cached --name-only | grep '\.md$')
node scripts/validate-frontmatter.js $(git diff --cached --name-only --diff-
filter=AM)

# Capability V3: legal lifecycle transitions
node scripts/validate-lifecycle.js $(git diff --cached --name-only --diff-
filter=M)

# Capability V5: reference integrity (быстрая проверка только изменённых)
node scripts/validate-references.js --changed-only $(git diff --cached --name-
only)
```

6.2 Pre-merge (CI job в `<project>.req`)

Полные проверки, которые медленны для pre-commit:

```
- name: Full reference validation (V5)
  run: node scripts/validate-references.js --all

- name: Coverage report regeneration (V4)
  run: node scripts/generate-coverage.js
  # commits as [coverage] bot user

- name: Drift detection (reconciliation, weekly)
  run: node scripts/detect-drift.js
  if: github.event.schedule == '0 0 * * 0'
```

6.3 Pre-merge (CI job в `<project>.src`)

```
- name: Submodule points to merged commit
  run: |
    cd requirements
    git fetch origin
    git merge-base --is-ancestor HEAD origin/main

- name: TC implementations pinned to the current set-version
  run: node scripts/validate-tc-version-pinning.js

- name: No Pass/Fail change without [test-spec-change]
  run: node scripts/validate-test-spec-changes.js
```

7. ТЗ workflow на git

7.1 Forward-workflow (проект с нуля)

Первичное создание оси требований из нового ТЗ:

1. **branch в `.req`:** `git checkout -b init/TZ-2026-001` в `<project>.req`.
2. **Регистрация ТЗ:** `tz/TZ-2026-001.md` (immutable после регистрации; зафиксировать подпись клиента и дату).
3. **Состязательный обзор ТЗ и — при находках — ADAPT:** обзор обязателен всегда, его вердикт фиксируется записью `ar/AR-001.md` (§7.4.1.2). При вердикте «findings present» создаётся `adapt/ADAPT-001.md` — Forward (интерпретация по каждому § ТЗ) + Backward (находки), standard/07; ADAPT — внутренний документ, клиенту он не выносится. При вердикте «no findings» ADAPT не создаётся, а BR / SR / SPEC несут `source.tz-section` + `source.adversarial-review-ref: AR-001` (§7.4.1.1).
4. **ACTZ:** находки, требующие решения клиента, выносятся протоколом `actz/ACTZ-001.md` — «Протокол уточнения ТЗ № 1» (§7.13). `status: draft → sent → signed`; подписи клиента и исполнителя фиксируются автором и отметкой времени (V6). После подписания каждая находка получает `decided-in: ACTZ-001 §M`.
5. **Подпись архитектора → QG-ADAPT-approve:** ADAPT в `approved` (подпись только архитектора, §7.5), `open-questions-count == 0`.
6. **Декомпозиция:** AI-агент выводит BR из approved ADAPT, затем SR (`source.adapt`) и SPEC (`source.adapt`; SR ссылается на них через `constrained-by[]`, §8.6.1), а также парные pos/neg TC.
7. **QG-0 approval** каждого артефакта (`draft → approved`); CI dry-run новых TC.
8. **PR в `.req` → merge;** bot regenerates REQUIREMENTS.md / COVERAGE.md / TEST-PLAN.md.
9. **Setup `.src`:** добавить submodule `requirements/ → <project>.req @ <commit>`, создать TR, реализовать, QG-2.
10. **AT перед испытаниями:** изолированный агент (другая модель, без доступа к `.src` и к внутренним артефактам `.req`) выводит `at/AT-NN.md` из итогового ТЗ — `tz/` +

подписанные `actz/` (§7.14). Прогон обновляет `ACCEPTANCE.md`; релизный гейт требует все AT зелёными (§10.4.3).

7.2 Delta-T3 workflow

Полная последовательность для применения нового delta-T3.

1. **branch в `.req`:** `git checkout -b change/TZ-2026-042` в `<project>.req`.
2. **Создание delta-T3, состязательный обзор, при находках — delta-ADAPT:** `tz/TZ-2026-042-delta.md` (текст delta); состязательный рецензент выносит вердикт, который фиксируется записью `ar/AR-NNN.md`. При вердикте «findings present» создаётся `adapt/ADAPT-NNN-delta.md` (forward интерпретация + backward findings, standard/07 §7.6); при вердикте «no findings» delta-ADAPT не создаётся, и затрагиваемые BR / SR / SPEC ссылаются на `source.tz-section: TZ-2026-042-delta` напрямую (§7.6.1). Решения клиента по находкам delta-T3 фиксируются новым `actz/ACTZ-NNN.md` (двусторонняя подпись); delta-ADAPT утверждается подписью архитектора прежде чем затронутые требования будут модифицированы. Итоговое ТЗ пересчитывается — от него же перегенерируются AT.
3. **Impact analysis:** AI-агент находит затронутые BR / SR / SPEC / TC; помечает TC как `obsolete-pending`.
4. **Update artefacts:** AI-агент обновляет / создаёт BR / SR / SPEC и парные TC (pos+neg) в той же ветке.
5. **Adversarial critic review:** на RENAR-5 — обязательно (другая AI-модель).
6. **CI dry-run:** новые TC должны запускаться без ошибок инфры.
7. **Finalize:** слияние черновика в `main` = версия комплекта `N.M+1` (тег `renar-vN.M+1`, запись версии), regenerate REQUIREMENTS.md / COVERAGE.md / TEST-PLAN.md (bot).
8. **PR в `.req` → QG-0 approval → merge.**
9. **branch в `.src`:** `git checkout -b change/TZ-2026-042` в `<project>.src`.
10. **Bump submodule:** `cd requirements && git pull && cd ..`.
11. **Create TR / tasks:** для новых / изменённых SR (через `/task` или специфичный для носителя tooling).
12. **PR в `.src` «bump requirements + tests + new tasks» → ревью → merge.**
13. **Development:** взять TR → реализовать → CI → бот заполняет `last-run` в TC.
14. **QG-2:** при зелёных TC на версии — runner дописывает запись верификации в запись версии; отдельного статуса у требования нет.

Каждый шаг кроме (1) и (9) автоматизирован нативно для носителя через скрипты или CI.

---

8. Migration notes: scripts/ модернизация

Существующие scripts/ — bash + node helpers из ранней эпохи проекта. Состояние модернизации на v1.0:

- ✅ **Legacy terms вычищены.** `req-branch.sh`, `req-finalize.sh`, `req-ai-instructions.md`, `req-use-template.sh` больше не используют устаревшие `INT-SR`, `AIC`, `UIC`, `tech-specs`, `ai-concepts`, `ui-concepts`. Closed list

актуальных типов — standard/04 §4.4 (BR/SR/TR + 11 SPEC). Остаточные упоминания в reference/01-glossary.md и reference/02-schemas.md являются legacy-mapping для проектов, мигрирующих с предыдущих версий.

- ✅ **Schema validator создан.** `scripts/validate-frontmatter.js` — Node ES-module, проверяет frontmatter всех `.md` в `standard/`, `guide/`, `reference/`, `core/`: required fields `title` / `order` / `lang ∈ {ru, en}`. Запуск: `node scripts/validate-frontmatter.js [--quiet]`; exit 0 если все валидны, exit 1 при первой ошибке. Источник схемы — reference/02-schemas.
- ✅ **ADAPT в forward-workflow.** Начальное создание (T3 → состязательный обзор → при находках ADAPT → BR) описано в §7.1 этого гайда; delta-workflow с delta-ADAPT — в §7.2 согласно standard/07 §7.6.
- ⏳ **Pre-commit hooks** (§6.1) пока частично разбросаны по `req-finalize.sh`. Целевое состояние — выделить в отдельные `scripts/validate-*.js`; `validate-frontmatter.js` — первый такой модуль.

Глава описывает **целевой набор скриптов** на выпуске v1.0; строки без отметки ✅ — работа для **фазы 8** (продолжение бэклога).

---

9. Common operations

Шорткаты для частых операций.

Операция	Команда
Найти SR по ID	`grep -r "^id: SR-12" sr/`
Найти TC, верифицирующий SR-12	`grep -rl "id: SR-12" tests/`
Найти orphan SR (без TC)	`node scripts/find-orphans.js sr`
Создать новый SR из шаблона	`cp library/templates/sr.md sr/SR-NN.md && $EDITOR sr/SR-NN.md`
Diff frontmatter за период	`git log --all --since=... -- sr/`
Текущая версия комплекта	`git describe --tags --match 'renar-v*'`
Список stale TC (last-run < current version)	`node scripts/list-stale-tc.js`

---

10. CI/CD integration patterns

10.1 Bot-commit conventions

Auto-generated артефакты коммитятся hooks носителя с conventional commit-message тегами для машинного парсинга:

Тег	Когда	Что коммитится

`[coverage]`	Post-merge в `.req`	Регенерация COVERAGE.md / REQUIREMENTS.md / TEST-PLAN.md
`[baseline-update]`	После approval PR с изменением эталона	Перегенерация PNG-эталонов для SPEC-UI
`[bump-req]`	Submodule bump в `.src`	Только submodule pointer + minimal metadata
`[reconcile]`	Reconciliation hook находит drift	Авто-fix очевидных рассогласований; flag для остальных
`[test-spec-change]`	Pass/Fail-критерий TC изменён	Manual; требует отдельный approval от reviewer

Hook носителя парсит commit message и применяет различные validation rules в зависимости от тега. `[coverage]` / `[bump-req]` / `[reconcile]` коммиты — от bot user; `[baseline-update]` / `[test-spec-change]` — от human + bot signature.

10.2 Bot-user setup

- Отдельный bot account (например `renar-bot@<org>`) с **write** permission в `<project>.req` и `<project>.src`.
- Bot commits signed (GPG / SSH commit signature).
- Bot user **не** может approve PR (separation of duties — утверждение всегда human).

10.3 branch protection

В обоих репо:

- `main` (или `master`) — protected. Push требует merged PR.
- Required status checks: schema validation (V2), reference integrity (V5), lifecycle validation (V3).
- Reviewers required: ≥ 1 для большинства; ≥ 2 для priority=must BR / SR / SPEC.

---

11. Перекрёстные ссылки

- standard/03-substrate-versioning — нормативные требования к носителю (capability V1-V6).
- reference/02-schemas — frontmatter schemas для validation hooks.
- 02-transition-guide — где этот guide вписывается в путь от pre-RENAR к RENAR-N.
- 04-document-store-substrate — informative обзор document-oriented store (альтернатива git).
- 07-failure-modes — что может пойти не так на git как носителе (схема обхода hooks, etc.).
- standard/10-lifecycle-qg — нормативные lifecycle переходы, которые validate-lifecycle hook должен enforce.

---

12. Open questions

- Стандартизировать ли минимальные реализации перехватов как **единое** спецификационное описание, на которое опираются и VCS, и document store?

- Submodule vs subtree: какие conditions делают subtree приемлемой альтернативой? Сейчас гайд однозначно за submodule.
- `[coverage]` bot user: best practice для signing / permission в multi-org проектах?
- Периодичность детекции дрейфа: weekly — хороший вариант по умолчанию, но что для high-velocity teams (> 50 PR/неделя)?

---

04. Носитель: document-oriented store (обзор)

> ***Информативное приложение.*** *Нормативные capability V1–V6 — standard/03. Практический пример на distributed VCS (git) — guide/03.*

Document-oriented store — носитель, где артефакты RENAR хранятся как версионированные документы: стабильный идентификатор, цепочка ревизий (revision token), атомарное обновление, API-шлюз для lifecycle-переходов и подписей.

RENAR формально **не привязан** к классу систем document store — нормативно задаются только возможности (**V1–V6**) (§3.3).

1. Когда выбирать document-oriented store

Подходит, если:

- нужно централизованное enterprise-хранилище требований для нескольких проектов;
- non-technical stakeholders работают через web UI, а не через VCS;
- федерация межпроектных ссылок и поиск — **ключевое** требование к продукту;

Не подходит, если:

- команда уже стандартизирована на git workflow и PR/MR review;
- нет ресурсов поддерживать сервер document store + search index + API gateway;
- нужны **полноценные** контрольные примеры (`TC`) как объекты в той же среде без отдельной настройки пользовательских типов документов (см. §9 — наличие `TC` нужно для декларируемого соответствия RENAR);

2. Сопоставление возможностей V1–V6 (обобщённо)

Возможность	Типичный механизм document store
V1 — неизменяемая история	Цепочка неизменяемых ревизий документа + стабильный `_id`
V2 — атомарная единица изменения	Одно целое приращение версии документа за шаг
V3 — сравнение и ревью (diff)	Поток утверждения через API и интерфейс; перехват прямых правок статуса
V4 — ветвление и слияние	Черновые документы либо ветви конфликтов (по возможностям продукта)
V5 — фиксация версии	Поле со ссылкой на ревизию в связанных артефактах
V6 — автор и отметка времени	Поля документа `author` + `updated_at` на каждую ревизию

Сравнение с распределённой VCS — §3.4 (таблица-пример; не является нормативным контрактом).

3. Миграция VCS ↔ document store

Миграция возможна, если в обеих реализациях носителя сохраняются:

- канонические идентификаторы артефактов (`BR` / `SR` / `SPEC` / `ADAPT`);
- связи трассируемости;
- состояния жизненного цикла по закрытому списку §10.

Процедура миграции — project-specific runbook; нормативный минимум — проверка V1–V6 до cutover (§3.8).

4. См. также

- 03-tool-guide-git.md — специфичный для носителя guide для distributed VCS (git)
- 03-substrate-versioning.md — нормативные V1–V6
- 02-schemas.md — canonical поля; нативное для носителя отображение — informative

05. Сравнение с SAFe

> *Mapping RENAR (стандарт **требований**) на SAFe 6.0 (стандарт **scaled agile координации**). Документы совместимы, но имеют разный scope: SAFe нормирует как команды координируют работу на масштабе; RENAR нормирует что собой представляет требование и как оно верифицируется. Эта глава — для команд, которые уже работают по SAFe (enterprise multi-team ART — типичный пример) и хотят сохранить SAFe ceremonies, добавив RENAR-артефакты как первичный источник истины о требованиях.*
>
> ***Предпосылки:** знакомство с RENAR Core — концептуальным обзором для человека-читателя и базовой SAFe 6.0 терминологией (Epic / Capability / Feature / Story, WSJF, PI, ART).*

1. Scope: что нормирует RENAR, что — SAFe

RENAR и SAFe пересекаются на уровне *артефактов работы* (Feature, Story), но решают разные задачи.

Аспект	SAFe 6.0	RENAR
Тип стандарта	Scaled Agile coordination framework	Стандарт инженерии требований
Первичный артефакт	Epic / Capability / Feature / Story	ТЗ → ADAPT → BR → SR → SPEC → TC
Что нормирует	Cadence, roles, ceremonies, flow	Schema, lifecycle, verifiability, drift control
Носитель артефактов	Выбор средства управления задачами (Jira / Rally / ADO / др.)	Без привязки к носителю; нормативно — возможности **V1–V6** (глоссарий §2.7)
Контрольные точки качества	Definition of Done на уровне Feature	`QG-0` (готовность к старту) + `QG-2` (проверено)
AI-нативность	Не нормирует процесс работы с ИИ	ИИ как полноценный участник (контрастная экспертиза *adversarial*, оценивающие сценарии, модели-судьи `judge`)

Ключевой принцип: SAFe и RENAR совместимы. SAFe говорит «как координировать команды на ART», RENAR — «что должно быть истинно про каждое требование, чтобы оно считалось `verified`». Feature в SAFe ≡ SR в RENAR — это **один и тот же артефакт**, описанный с разных сторон.

2. Главная таблица соответствия

SAFe artefact	RENAR artefact	Где живёт	Owner	Acceptance criteria
Strategic Theme	(вне scope RENAR — корпоративная стратегия)	стратегические доки	Executive	бизнес-уровень
Portfolio Epic	Группа BR одной системы	`<system>.req/br/` aggregated	Lean Portfolio Mgmt	Все BR группы входят в верифицированную версию комплекта
Capability	BR подсистемы (если у подсистемы свой stakeholder)	`<subsystem>.req/br/`	Solution Architect	Все BR подсистемы входят в верифицированную версию комплекта
Program Epic	Опционально — крупная инициатива внутри ART, aggregated SR	`<system>.req/sr/` aggregated	Product Manager	Заданная outcome metric достигнута
Feature	SR (или связанная группа SR)	`<subsystem>.req/sr/`	Product Owner	TC из `verified-by` зелёные при `last-run.set-version` = текущей версии комплекта (QG-2)
Story	TR (Task Requirement)	`<subsystem>.req/tr/` или task tracker (Jira, Linear, GitLab Issues)	Команда	Все AC выполнены, evidence зафиксирован, code merged
Enabler Epic	SPEC-AI / SPEC-ARCH / SPEC-OPS	`<system>.req/specs/`	Architect	Eval-метрики в пределах thresholds; эталон зафиксирован
Spike	Исследовательский TR + Decision в decision log	`<subsystem>.req/tr/` + decision log	Команда	Decision записан и связан с originating SR
Defect	TR с `parent: SR-NN` (нарушенный SR)	task tracker + auto-derived `children[]` в SR	Команда	Negative TC проходит, regression test добавлен

Запрещённые соответствия: INT-SR — устаревший термин (§4.14.1 Terms), заменён `SR` с `constrained-by: [SPEC-INT-N]`. См. §6 ниже.

---

3. WSJF prioritization для RENAR

SAFe использует **WSJF (Weighted Shortest Job First)** как приоритизационный framework. RENAR адаптирует его для уровня BR / SR.

3.1 Формула

```
WSJF = (User-Business Value + Time Criticality + Risk Reduction & Opportunity
Enablement) / Job Size
```

Каждый компонент оценивается по шкале 1-20 (modified Fibonacci); WSJF — относительная метрика, имеет смысл только при сравнении BR/SR внутри одного backlog.

3.2 Расширение frontmatter BR

```yaml
---
id: BR-12
title: "Сократить время регистрации сотрудников до < 2 минут"
priority: must
prioritization:
  framework: WSJF
  components:
    user-business-value: 10
    time-criticality: 8
    risk-reduction-opportunity: 6
    job-size: 5
  wsjf-score: 4.8               # auto-calculated: (10+8+6)/5
  prioritized-at: 2026-05-03
  prioritized-by: "@product-owner"
---
```

Поле `prioritization` — опциональное в Core RENAR, обязательное на ART, который применяет SAFe.

3.3 Когда применяется

- **Обязательно** для BR с `priority: must` в проектах, координируемых через ART.
- **Рекомендуется** для всех BR в backlog, который проходит PI Planning.
- **Не применяется** для SR — SR наследует приоритет родительского BR. Перешафливание SR внутри одного BR — задача Product Owner на decomposition, не WSJF.

3.4 Альтернативы

Если команда не использует SAFe / WSJF — RENAR допускает другие frameworks:

- **MoSCoW** (Must / Should / Could / Won't) — простейший, для маленьких проектов. Уже есть как `priority` enum в RENAR Core.
- **RICE** (Reach × Impact × Confidence / Effort) — для product-driven команд (типично B2C).

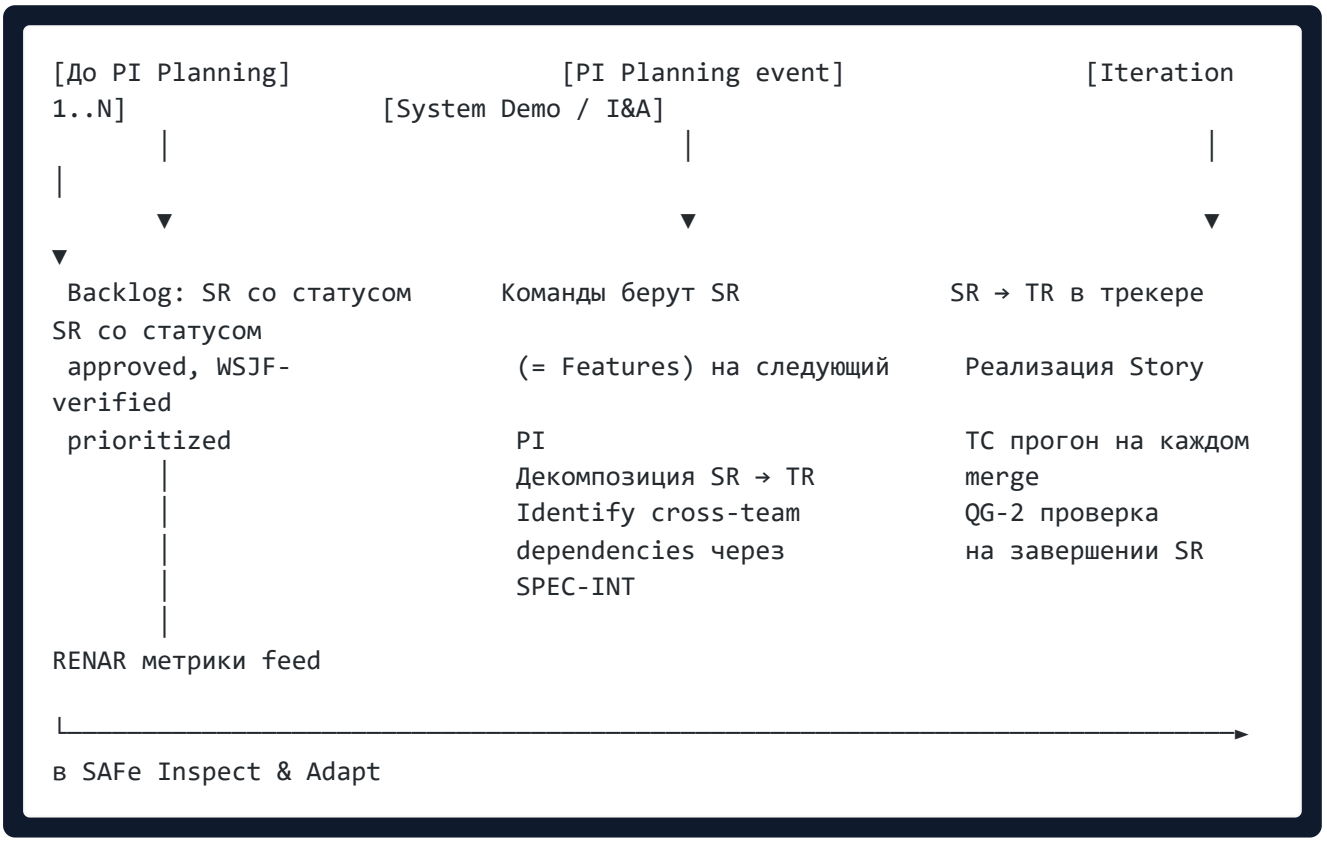
RENAR нормирует **поле и его schema**; выбор framework — на проекте. См. также reference/02-schemas.md для допустимых значений `prioritization.framework`.

4. PI Planning интеграция

4.1 Что такое PI

Program Increment (PI) — фиксированный отрезок времени (обычно 8-12 недель) в SAFe, в течение которого ART коммитится на набор Features из общего backlog. PI Planning — двух- или трёхдневный event перед каждым PI, где команды совместно фиксируют commitment.

4.2 Где RENAR-артефакты попадают в PI flow



4.3 Артефакты RENAR в каждой ceremony

SAFe ceremony	RENAR input	RENAR output
Backlog refinement	BR / SR в черновике или в утверждённой версии комплекта	Уточнённый ADAPT, обновлённый WSJF
PI Planning	WSJF-sorted SR backlog, ADAPT-доки	Commitment на SR в PI; SPEC-INT для cross-team
Iteration Planning	SR + декомпозированные TR	TR в `approved`
System Demo	SR верифицированной версии комплекта	Evidence из записи верификации и TC last-run
Inspect & Adapt	Метрики RDLT, Coverage Velocity, Hallucination Rate	Корректировки backlog / процесса

5. ART координация и роли

5.1 SAFe роли ↔ RENAR ответственности

SAFe role	RENAR ответственность
RTE (`Release Train Engineer`)	Координация cross-team dependencies через SPEC-INT; владелец PI Objectives ↔ RENAR метрик mapping
Product Manager	Owner портфельных Epic = групп BR на уровне системы
Product Owner	Owner Feature = SR на уровне команды; accountable за QG-0 (утверждение версии) и QG-2 (верификация версии)
System Architect	Owner SPEC-* (особенно SPEC-ARCH, SPEC-INT); consulted при декомпозиции BR → SR
Tech Lead	Accountable за QG-2 на уровне SR; владеет TR-backlog команды
Business Owner	Approver BR / ADAPT с бизнес-стороны
Solution Architect	Owner BR на уровне подсистемы (когда подсистема имеет свой stakeholder)
Scrum Master	Owner ceremonies; не владеет RENAR-артефактами напрямую

5.2 Cross-team координация

В типичной SAFe-организации с несколькими командами в одном ART:

- **Каждая команда** владеет своими SR / TR в `<subsystem>.req/`.
- **RTE / Solution Architect** владеют SPEC-INT в `<system>.req/specs/int/` — общими integration contracts между подсистемами.
- **Изменение SPEC-INT** требует cross-team согласования (QG-0 от всех затронутых команд).

Правило: ART-уровневые роли (RTE) **не редактируют** SR в подсистемах напрямую. Координация — через SPEC-INT, который явно `constrained-by` для каждой затронутой SR.

6. Cross-team dependencies через SPEC-INT

Когда Feature в одной подсистеме блокирует / зависит от другой:

6.1 SR с зависимостью

```
# В <subsystem-a>.req/sr/SR-05.md
---
id: SR-05
parent: BR-12
title: "Регистрация пользователя через корпоративный email"
constrained-by:
  - id: SPEC-INT-01
```

```
    ref: "specs/int/SPEC-INT-01-auth-handshake.md"
verified-by:
  - TC-23
  - TC-24
---
```

6.2 SPEC-INT как контракт

```
# B <system>.req/specs/int/SPEC-INT-01-auth-handshake.md
---
id: SPEC-INT-01
type: SPEC-INT
title: "Auth handshake между Subsystem A и Subsystem B"
version: 1.2
participants:
  - subsystem: "subsystem-a"
    role: "client"
  - subsystem: "subsystem-b"
    role: "provider"
verified-by:
  - TC-INT-01    # contract test
---
```

6.3 Breaking changes в SPEC-INT

Любое изменение `SPEC-INT.version` с breaking-семантикой (§4.11 Drift классы) требует:

1. ADAPT-уровневое обсуждение (зачем меняем).
2. Согласование от всех `participants` (QG-0 от каждой команды).
3. Migration plan для existing implementors (как старые SR останутся valid или будут адаптированы).

RTE **обязан** проверять SPEC-INT consistency между подсистемами регулярно (обычно раз в спринт) — это часть Inspect & Adapt и feed в conformance self-assessment (§13 Conformance).

---

7. Definition of Done на каждом уровне иерархии

DoD — **формальные** условия, проверяемые автоматически (hooks носителя). На каждом уровне иерархии — свой DoD.

Level	RENAR artefact	DoD criterion
Strategic Theme	—	(вне scope RENAR)
Portfolio Epic	Группа BR	Все BR группы — в верифицированной версии комплекта, KPI impact подтверждён через outcome metric

Capability	BR подсистемы	Все BR — в верифицированной версии; outcome metric измерен
Feature	SR	QG-2 passed: все TC из `verified-by` имеют `last-run.result = pass` при `last-run.set-version` = версии комплекта
Story	TR	Все AC выполнены, evidence зафиксирован, code merged, automated test exists
Enabler	SPEC-AI / SPEC-ARCH / SPEC-OPS	Eval-runs прошли пороги (для SPEC-AI); эталон зафиксирован (для всех)

Ключевое: на уровне Feature DoD в SAFe **совпадает** с QG-2 в RENAR. Нет двух разных «definition of done» — это один и тот же gate, описанный с разных сторон.

8. Built-in Quality ↔ RENAR mechanisms

SAFe принцип Built-in Quality: качество встроено в процесс, не ad-hoc. RENAR реализует Built-in Quality через нормативные механизмы:

SAFe Built-in Quality practice	RENAR mechanism
Continuous Integration	hooks носителя + CI на каждый change в требованиях (§13 Conformance); reconciliation hook (drift detection, §4.11)
Test-First	TC-нормы создаются **до** реализации и утверждаются в той же версии комплекта, что и SR; QG-0 комплекта требует пару TC-норм на каждое утверждение
Refactoring	Continuous reconciliation hook (§2.4.2); обнаруживает дрейф между требованиями и кодом
Pairing / Mobbing	AI-генератор + AI-критик (§5.2 Roles) — pair generation/review как нормативная роль
Definition of Done	QG-2 как формальный gate, проверяемый автоматически (§10 Lifecycle и QG)
Version Control	Возможность **V1** (неизменяемая история) без привязки к классу сред хранения
Automation	Capability V2-V6 — все верификации автоматизируемы

RENAR **не предписывает** instrument (Jenkins / GitLab CI / GitHub Actions / Tekton) — только **что** должно проверяться (capability), не **как**.

9. RACI lifecycle артефакта (с SAFe ролями)

Полный жизненный цикл RENAR-артефакта с распределением ответственности по SAFe ролям.

Активность	Responsible	Accountable	Consulted	Informed
Импорт ТЗ	AI-агент	System Architect	Business Owner	Команда, RTE

Декомпозиция T3 → ADAPT	AI-генератор	System Architect	Stakeholder, AI-критик	RTE, Команда
Декомпозиция → BR	AI-генератор	Product Owner	Business Owner, AI-критик	RTE, Команда
WSJF prioritization	Product Manager	Product Owner	RTE, Stakeholder	Команда
Декомпозиция BR → SR	AI-генератор	System Architect	AI-критик	Команда, RTE
Декомпозиция SR → SPEC-*	System Architect	System Architect	AI-критик, Tech Lead	Команда
Генерация TC	AI-агент	Test Architect	—	Команда
QG-0 approval	System Architect	Tech Lead	AI-критик	Business Owner, RTE
Выбор SR на PI	Product Owner	RTE	Команда (capacity)	Stakeholder
Декомпозиция SR → TR	Команда	Product Owner	Tech Lead	RTE
Реализация TR	Разработчик	Tech Lead	—	Команда
Прогон TC (automated)	hook носителя	—	—	Команда
QG-2 (версия верифицирована)	System Architect	Tech Lead	—	Business Owner, RTE
System Demo	Команда + RTE	Product Owner	Stakeholder	RTE, Executive
Утверждение delta-T3	System Architect + Stakeholder	Product Owner	AI-impact analysis, RTE	Команда
Spot-check 5 TC (audit)	System Architect	—	—	RTE
Reconciliation MR	AI-агент-reconciler	System Architect	—	Команда

10. PI Objectives ↔ RENAR метрики

PI Objectives — SMART outcomes на квартал. RENAR-метрики (§12 Metrics, reference/02-schemas.md) feed в PI Objectives:

PI Objective (example)	RENAR метрика
«Сократить время от подписания T3 до первого commit до < 2 дней»	**RDLT** (Requirement Decomposition Lead Time)
«Достичь Coverage Velocity ≥ 60% в спринт»	**Coverage Velocity** (% SR с `status: verified` per sprint)

«Снизить Hallucination Rate в новых требованиях до < 2%»	**Hallucination Rate** (% AI-сгенерированных утверждений, отклонённых на review)
«0 disputed requirements на acceptance в этом PI»	**Dispute Rate at Acceptance**
«Drift detection — все SR consistent с code в течение 24 часов после merge»	**Reconciliation Findings per Week** (§12.3.10)

Это превращает RENAR из «standard for documents» в **measurable contribution** к ART success. Метрики feed автоматически в Inspect & Adapt; RTE использует их при ретроспективе на завершении PI.

---

11. Что из SAFe оставить, что заменить

Для команд, переходящих на RENAR с уже работающего SAFe-процесса:

11.1 Оставить как есть

- **Cadence** (PI, Iterations) — RENAR не нормирует время; используйте свой ритм.
- **Ceremonies** (PI Planning, System Demo, I&A, Daily Standup) — RENAR-артефакты прозрачно вписываются в эти events.
- **WSJF** — оставить для приоритизации BR / SR; вписывается в `prioritization.framework: WSJF`.
- **ART, RTE, Scrum Master** — структура и роли сохраняются.
- **PI Objectives** — оставить; mapping на RENAR-метрики (§10) делает их измеримыми.

11.2 Заменить RENAR-эквивалентом

- **Feature description в Jira / Rally / ADO** → SR со полным frontmatter в `<subsystem>.req/sr/`. Tracker-запись становится **зеркалом** RENAR-артефакта, не первичным источником.
- **Acceptance criteria в Feature** → `verified-by: [TC-NN]` с автоматически проверяемыми TC.
- **Definition of Done на Feature** → QG-2 (нет двух разных DoD — это один gate).
- **Integration agreements между командами** → SPEC-INT (заменяет INT-SR из старых SAFe-реализаций).
- **Architecture Decision Records (ADR)** → SPEC-ARCH / SPEC-AI / SPEC-OPS (один из 12 типов SPEC, §4.4).

11.3 Добавить (новое в RENAR)

- **ADAPT** — мостовой артефакт между T3 и иерархией требований. В SAFe нет прямого аналога; создаётся **реактивно** (standard/07 §7.4.1) — только когда состязательный рецензент вынес вердикт «findings present». При вердикте «no findings» ADAPT не создаётся, и BR / SR / SPEC выводятся из T3 напрямую через `source.tz-section` + `source.adversarial-review-ref`. Обязателен не ADAPT, а сам состязательный обзор.

- **AI-критик роль** — адверсариальная проверка AI-генерации. В SAFe не нормирована, в RENAR Core — обязательна для всех AI-сгенерированных артефактов.
- **Reconciliation (drift detection)** — непрерывная сверка требований с реализацией (опирается на V5 — сквозную фиксацию версии). В SAFe — manual reconciliation, в RENAR — нормативный механизм.

---

12. Negative: чего эта глава не утверждает

- **RENAR — не замена SAFe.** RENAR нормирует *требования*, SAFe — *координацию работы*. Команда может работать по RENAR без SAFe (маленький проект, одна команда) или с SAFe (enterprise multi-team ART scope).
- **RENAR — не стандарт планирования.** Cadence, capacity planning, velocity tracking — все за пределами scope. RENAR говорит «что должно быть истинно про требование», не «когда команда должна его доделать».
- **RENAR не запрещает другие frameworks.** WSJF, MoSCoW, RICE — все совместимы. RENAR фиксирует **поле** `prioritization.framework`, не выбор framework.
- **RENAR не нормирует Jira / Rally / ADO workflow.** tracker-уровневая реализация — деталь носителя; нормативный источник — `<subsystem>.req/`.
- **RENAR не предписывает PI как обязательный.** Команда может работать по непрерывному flow без PI; в этом случае разделы 4 и 10 этой главы становятся informative.

---

13. Resolved decisions для v1.0

- `priority: must` **НЕ требует WSJF score даже в SAFe-проектах.** Per §12 этой главы: RENAR фиксирует `prioritization.framework`, не предписывает выбор framework. `priority: must` — это RENAR MoSCoW marker (reference/02-schemas), независимый от WSJF. WSJF — оптимизация SAFe-команд, не нормативное требование RENAR.
- **Feature/Capability/Story mapping — специфично для носителя.** RENAR нормирует closed list типов требований (BR/SR/TR + 11 SPEC) и не вводит SAFe Feature-уровни. Проекты, применяющие SAFe, фиксируют mapping в нативном для носителя `safe-mapping/` manifest (см. reference/02-schemas — informative extension).
- **PI Objectives — informative, вне scope RENAR.** PI — SAFe coordination artifact, RENAR не нормирует. Если команда хочет traceability, рекомендуется informative `cross-link.pi-objective-id` поле в BR-frontmatter — это не conformance-gating, но облегчает RTE-side queries.
- **Inspect & Adapt: метрики нативно для носителя автоматизированы.** Per §10.13 Logging + §12 — носитель обязан нативно для носителя выставлять COVERAGE / audit-trail. RTE использует те же данные через query API, manual extraction — anti-pattern (drift).

13.1 Отложено на v1.1 (бэклог фазы 8)

- **Эвристика ИИ для оценки поля WSJF Job Size** (шкала 1–20 по числу AC, сложности `TC` и площади кода). В v1.0 не нормируется; специфично для связки SAFe–RENAR. Расширенный informative mapping — reference/11.

---

14. Связь с другими главами

- 00-quickstart — базовый цикл RENAR без SAFe-надстройки.
- 01-walkthrough — полный example на small-scale проекте (без PI Planning).
- reference/01-glossary — точная семантика BR / SR / SPEC / TR / TC.
- reference/02-schemas — frontmatter schemas, включая `prioritization`.
- standard/04-terms — нормативные определения; mapping на SAFe закреплён в §4.13.
- standard/08-specifications — closed list 12 типов SPEC, включая SPEC-INT и SPEC-UC.

06. Соответствие

> *RENAR — стандарт инженерии требований; не compliance-стандарт сам по себе. Но RENAR предоставляет **инфраструктуру traceability**, которая делает соответствие другим стандартам (ISO 27001, GDPR, ФЗ-152, EU AI Act и др.) проверяемым автоматически. Эта глава — mapping артефактов RENAR на 8 ключевых compliance фреймворков + чек-листы самооценки + список auto-generated artifacts для аудитора.*
>
> ***Предпосылки:*** *RENAR Core, reference/02-schemas.md, reference/03-ai-risk-register.md.*

1. Принципы compliance в RENAR

1.1 Compliance frontmatter. Каждое требование с регуляторным обоснованием **обязано** содержать поле `compliance` во `frontmatter` — трассировать соответствие во внешней таблице без связи с артефактом нельзя. Поле повторяемое (одно требование может закрывать controls нескольких стандартов):

```yaml
compliance:
  - { standard: "ISO 27001:2022", control: "A.5.34", rationale: "Privacy and
protection of PII" }
  - { standard: "GDPR", article: "Art.32", rationale: "Security of processing" }
  - { standard: "ФЗ-152", article: "ст.19", rationale: "Меры по обеспечению
безопасности ПДн" }
```

1.2 Data classification. Каждое BR/SR, оперирующее данными, обязано декларировать классификацию. При `contains-pii: true` автоматически становятся обязательными SR-encryption + SR-audit-log + SR-erasure.

```yaml
data-classification:
  contains-pii: true              # GDPR / ФЗ-152 trigger
  contains-financial: false       # PCI-DSS trigger
  contains-health: false          # HIPAA / ФЗ-152 спец. категории
  contains-children-data: false   # COPPA trigger
  retention-days: 1095
  data-residency: ["RU", "EU"]
```

1.3 Traceability как audit foundation. Цепочка `BR → SR → SPEC → TC → реализация → CI run` — это и есть журнал аудита. Аудитор открывает coverage report и видит: какие требования закрывают control; какие TC верифицируют; `last-run.date`; `last-run.set-version` (версия комплекта, на которой прошёл прогон). Время аудита: 1-2 дня вместо 2-3 недель. Reconciliation hook (drift detection) гарантирует, что evidence не «протух» между аудитами.

2. ISO/IEC 27001:2022 — Information Security

ISO 27001:2022 Annex A control	RENAR-артефакт
A.5.7 Threat intelligence	SPEC-SEC с threat model; SR с `compliance: A.5.7`
A.5.8 Information security in project management	Сам RENAR как process compliance
A.5.34 Privacy and protection of PII	SR с шифрованием + `data-classification.contains-pii: true`
A.6.3 Information security awareness	Onboarding процесс включает чтение RENAR
A.8.1 User endpoint devices	SR с device requirements (если в scope)
A.8.5 Secure authentication	SR-AUTH-* + SPEC-SEC с authentication flow
A.8.7 Protection against malware	SR + adversarial review (AI-критик)
A.8.10 Information deletion	SR с deletion logic + `retention-days`
A.8.16 Monitoring activities	SR с logging + SPEC-OPS audit-log
A.8.24 Use of cryptography	SR с явным crypto algorithm + ISO 25010 Security
A.8.25 Secure development life cycle	SENAR + RENAR как evidence
A.8.28 Secure coding	TC с security checks; SPEC-SEC с STRIDE coverage
A.12.1.2 Change management (наследие 27001:2013)	Delta-T3 + Impact Analysis (§7.6)

Результат для аудита. Auto-generates conformance report: scope (BR/SR/TC counts с `compliance.iso27001`) + Coverage by Annex A (control ↔ mapped SR ↔ status). Нормативный механизм — capability V4 reporting from versioned artifacts.

3. GDPR (Regulation EU 2016/679)

GDPR Article	RENAR-артефакт
Art.5 Principles	BR явно указывает lawful basis
Art.6 Lawfulness	BR с `gdpr.lawful-basis: consent / contract / legal-obligation / vital-interests / public-task / legitimate-interests`
Art.7 Conditions for consent	SR с consent management workflow
Art.13 Information to data subject	SPEC-UI с privacy notice экраном
Art.15 Right of access	SR с export-user-data
Art.16 Right to rectification	SR с edit-user-profile

Art.17 Right to erasure	SR с delete-user-data + propagation на linked entities
Art.18 Right to restriction	SR с suspend-processing
Art.20 Right to data portability	SR с export в machine-readable формате
Art.25 Data protection by design and by default	`data-classification` обязателен на BR-уровне
Art.30 Records of processing activities	Auto-generated из BR с `contains-pii`
Art.32 Security of processing	SR с encryption + access control
Art.33 Notification of personal data breach	SR с breach notification workflow + 72h timer
Art.35 DPIA	Для high-risk — отдельный `dpia/<feature>.md`

GDPR frontmatter:

```
gdpr:
  data-categories: ["identification: email, name", "contact: phone, address",
"behavioral: usage logs"]
  lawful-basis: contract              # Art.6
  retention-period-days: 1095
  cross-border-transfer: false        # true → обязательны SCCs или adequacy
decision
  dpia-required: false                # true → link на DPIA
  data-subject-rights: [access, rectification, erasure, portability]   #
Art.15/16/17/20
```

DPIA. Для high-risk processing — `<system>.req/dpia/DPIA-NN-<slug>.md` с frontmatter (`id`, `type: DPIA`, `gdpr-article: 35`, `related-br[]`, `risks-identified`, `mitigations`) и секциями: процесс обработки, необходимость, риски, меры снижения, согласование с DPO.

4. ФЗ-152 «О персональных данных» (РФ)

ФЗ-152 Статья	RENAR-артефакт
ст.5 Принципы обработки	BR с целевым назначением (`business-context.business-goal`)
ст.6 Условия обработки	BR с `lawful-basis`
ст.9 Согласие на обработку	SR с consent management
ст.13.1 Хранение в РФ	`data-classification.data-residency: ["RU"]` для российских клиентов

ст.14 Доступ к информации	SR с export-user-data
ст.15 Право на уточнение	SR с edit-user-profile
ст.16 Удаление	SR с delete-user-data
ст.18 Обязанности оператора	Журнал аудита через RENAR traceability
ст.18.1 Локализация ПДн граждан РФ	`data-residency` обязательно
ст.19 Меры защиты	SR с encryption + access control + ISO 25010 Security
ст.21 Уведомление о нарушении	SR с breach notification workflow
ст.22 Уведомление РКН	operational, вне scope RENAR

Контроль резидентности данных. Для проектов с российскими клиентами `data-residency` обязательно. Hook носителя проверяет: при `data-residency: ["RU"]` все upstream SR (хранение, бэкапы, репликация) должны иметь evidence хранения в РФ; добавление зарубежной юрисдикции для RU-резидентного BR → блок на QG-0.

5. EU AI Act (Regulation EU 2024/1689)

AI Act EU классифицирует AI-системы по уровню риска. RENAR обязывает явно указывать класс:

```
ai-act:
  risk-class: limited                # prohibited | high | limited | minimal
  rationale: "Generative AI for document drafting; no autonomous decisions
affecting users' legal rights"
  high-risk-domain: false            # true → требуется conformity assessment
  general-purpose-ai: true           # GPAI — Claude / GPT и т.д.
```

High-risk AI requirements (Art.9-15) — для класса `high` (financial scoring, hiring, public services, медицинская диагностика):

AI Act требование	RENAR-артефакт
Art.9 Risk management system	SPEC-AI + AI risk register (reference/03)
Art.10 Data and data governance	`eval-datasets/` с provenance + spot-check
Art.11 Technical documentation	SPEC-AI + ISO/IEC 5338 conformance (§7)
Art.12 Record-keeping	tool_event audit + `ai-provenance`
Art.13 Transparency to users	SPEC-UI с обозначением AI-обработки
Art.14 Human oversight	One-click утверждение + spot-check на QG-0 / QG-2
Art.15 Accuracy, robustness, cybersecurity	Eval-tests (`tc-type: eval`) + adversarial review

GPAI обязательства (Art.51-55). Если используется General-Purpose AI Model (Claude, GPT, Gemini): `ai-provenance` во frontmatter любого AI-сгенерированного артефакта (model id, version, prompt hash); технический документ модели (если GPAI с systemic risk) — внешний документ + reference из SPEC-AI.

6. NIST AI RMF 1.0 (US-юрисдикция)

NIST AI RMF Function	RENAR-механизм
Govern	RENAR + роли SENAR + compliance frontmatter
Map	BR с `business-context`, `data-classification`, `ai-act.risk-class`
Measure	Eval-тесты с metric thresholds + RENAR-метрики (Hallucination Rate, Coverage Velocity)
Manage	Lifecycle с QG + reconciliation hook + AI risk register

RMF-специфичные расширения (опционально для US-проектов; не противоречит ISO/IEC 23894 §7):

```
nist-ai-rmf:
  applicable: true
  govern: { role: "AI Governance Lead", policy-link: "..." }
  map: { use-case-category: "content-generation", affected-stakeholders:
["clients", "internal-team"] }
  measure: { metrics-tracked: ["accuracy", "fairness", "robustness"], baseline-
run-id: "eval-2026-04-01" }
  manage: { risk-tolerance: "low", incident-response-plan: "<link>" }
```

7. ISO/IEC 23894:2023 — AI Risk Management

Guidance по AI risk management. Не сертифицирует, но даёт structured framework для управления рисками AI-систем. Совместим с NIST AI RMF и AI Act EU.

ISO/IEC 23894 раздел	RENAR-артефакт
§6.4.1 Risk identification	AI risk register (reference/03)
§6.4.2 Risk analysis	SPEC-AI с risk assessment секцией
§6.4.3 Risk evaluation	Threshold для каждого риска в `eval-tests`
§6.4.4 Risk treatment	SR-mitigations + post-mitigation evaluation
§6.4.5 Communication and consultation	RACI matrix (05-safe-comparison §9)
§6.4.6 Monitoring and review	Reconciliation hook (drift detection)

Критерии соответствия: каждый AI-сгенерированный артефакт имеет `ai-provenance`; для каждого AI use-case в SPEC-AI задокументированы identified risks (hallucination, bias, robustness), mitigations (eval thresholds, adversarial review, human-in-the-loop), residual risks; risk register обновляется при изменении SPEC-AI (reconciliation ловит drift).

8. ISO/IEC 5338:2023 — AI System Life Cycle Processes

Extension к ISO/IEC/IEEE 12207. Удобный фреймворк для AI Act Art.11 technical documentation.

ISO/IEC 5338 process	RENAR-этап
Stakeholder needs and requirements definition	T3 + ACTZ (протоколы уточнения) + ADAPT + BR
Architecture definition	SPEC-ARCH + SPEC-AI
Design definition	SPEC-AI (model card, prompt design, RAG)
Implementation	TR + реализация
Verification	TC (`tc-type: eval` для AI)
Validation	AT — приёмочные тесты от итогового T3 (§9.19) + spot-check
Operation	Reconciliation hook (drift detection, §4.11)
Maintenance	Delta-T3 + ACTZ + ADAPT iteration
Disposal	Retirement workflow (вне scope Core RENAR)

Доказательная база соответствия: SPEC-AI для каждой AI use-case с полной model card; eval-tests привязаны к SPEC-AI через `verifies[]`; `ai-provenance` зафиксирован; drift detection активен (V6).

9. PCI-DSS v4.0 — Payment Card Industry Data Security Standard

Если проект обрабатывает данные платёжных карт (CHD):

PCI-DSS v4 requirement	RENAR-артефакт
Req.1 Network security controls	SPEC-OPS с network segmentation
Req.3 Protect stored CHD	`contains-financial: true` + SR с encryption-at-rest
Req.4 Encrypt CHD transmission	SR с TLS + SPEC-API requirements
Req.6 Develop secure systems	RENAR + SENAR как process compliance
Req.7 Restrict access by need to know	SR с RBAC + SPEC-SEC access control matrix
Req.8 Authenticate access	SR-AUTH-* + MFA где обязательно
Req.10 Log and monitor access	SR с журналом аудита + SPEC-OPS observability

Req.11 Test security regularly	TC `tc-type: security` + pen-test (вне RENAR scope)
Req.12 Information security policy	RENAR + SENAR как documented policy

Минимизация объёма CHD. hook носителя проверяет, что новые SR с `contains-financial: true` явно обосновывают необходимость хранения CHD — без обоснования требование останавливается на QG-0.

---

10. Чек-листы самооценки

Короткие yes/no чек-листы для compliance-команд. Полный печатный kit для RENAR manifest — reference/08.

ISO 27001:2022 — все BR с `contains-pii: true` имеют SR закрывающий A.5.34; SoA задокументирован; каждый in-scope control имеет ≥1 verifying SR; coverage report зелёный; аудитор проходит от control до passing TC за <5 минут.

GDPR — все PII в Records of Processing (auto-generated); lawful basis указан per processing activity; data subject rights Art.15-22 verifiable через SR + TC; DPIA выполнен для high-risk; cross-border transfers обоснованы (SCCs/adequacy).

ФЗ-152 — требования с PII граждан РФ имеют `data-residency: ["RU"]`; hook носителя проверяет upstream SR хранения; согласие реализовано в SR с TC evidence; меры защиты ст.19 покрыты SR с passing TC.

EU AI Act — каждый SPEC-AI имеет `ai-act.risk-class`; для `high`: Art.9-15 evidence в носителе; human oversight на QG-0/QG-2 + spot-check; technical documentation auto-generated; `ai-provenance` для GPAI-компонент.

NIST AI RMF — все 4 функции (Govern/Map/Measure/Manage) имеют доказательную базу; AI Governance Lead определён в roles; эталон eval зафиксирован; incident response plan связан со SPEC-AI.

ISO/IEC 23894 — AI risk register заполнен и связан со SPEC-AI; каждый риск имеет mitigation в SR; residual risks accepted владельцем; V6 активна.

ISO/IEC 5338 — SPEC-AI для каждой AI use-case с model card; eval-tests привязаны через `verifies[]`; `ai-provenance` зафиксирован; V6 активна.

PCI-DSS — SR с `contains-financial: true` имеют encryption (at-rest + in-transit); CHD scope минимизирован; RBAC в SPEC-SEC; SR журнала аудита покрывает все CHD-touching flows; security TC прогоняются регулярно.

10.9 Самооценка RENAR-conformance (за час)

Проект декларирует **собственный** уровень RENAR-conformance через manifest `RENAR-CONFORMANCE.yaml` (standard/13 §13.4). Быстрый чек-лист (yes/no, один проход):

- Носитель требований обеспечивает все V1–V6 (§11.4.1 — Notion/Google Docs не подходят).
- На каждое T3 выпущена AR (запись состязательного обзора); каждый ADAPT, порождённый вердиктом «findings present», — в статусе approved с **подписью архитектора** (§7.5). ADAPT может не быть вовсе (вердикт «no findings») либо быть несколько (находки на разных стадиях) — обе ситуации штатны.

- Обязательная подпись клиента под ADAPT **не объявлена**: клиентские обязательства несёт ACTZ. Требовать подпись клиента на ADAPT можно только как `declared-stricter`, но не как базовое соответствие (§13.3.3).
- Каждая находка в статусе `resolved` несёт `decided-in` на пункт **подписанного** ACTZ; ни один подписанный ACTZ не остался неотражённым в ADAPT (§7.4.5, §7.13).
- Приёмка идёт против **итогового ТЗ** (начальное ТЗ + все подписанные ACTZ, §7.14), а не против ADAPT.
- AT выведены изолированным агентом: модель отлична от модели основного агента, доступа к ADAPT / BR / SR / SPEC / TC / коду не было; `tz-version` каждого AT совпадает с действующей редакцией итогового ТЗ; все AT в `passing` перед предъявлением (§9.19, §10.4.3).
- Ни один TC не засчитан как доказательство без красной истории; исключение — TC класса `implementation-originated`, где компенсацией служит убитый мутант (§9.18.2).
- Требования класса `implementation-originated` описывают только внутренние технические детали: наблюдаемое клиентом поведение через этот класс не легализовано (§6.13.2).
- Все 12 типов SPEC поддерживаются нативно (declaration «type не используется» допустима, «не поддерживается» — нет).
- Каждое нормативное утверждение, покрытое TC, имеет парный negative TC.
- QG-0 / QG-1 / QG-2 объявлены `required`.
- Манифест содержит `renar-version` + `senar-version` + `level` + подтверждение mandatory clauses (reference/08 §2).
- `assessment-date` свежее, `next-assessment-due` не просрочен.

Все 13 — `yes` → проект conformant к заявленному `level`. Хотя бы один `no` → manifest не выпускается (§13.5.2).

Четыре пункта списка — 3, 6, 7 и 8 — держат одну мысль: **обязательство перед клиентом не может возникнуть ни из инженерного документа, который клиент не читал, ни из кода, ни из теста, написанного тем же агентом и той же моделью, что и реализация.** Источники у них разные, и стоит их назвать поимённо: подпись клиента под ADAPT не объявлена — §13.3.3 и §7.5; изоляция генератора AT — §9.19.2 с точкой контроля §10.11.1; красная история — §9.18.2; границы класса `implementation-originated` — §6.13.2. Из четырёх только первый входит в закрытый перечень обязательных положений §13.3; остальные три объявлены обязательными в своих главах.

Заполненный пример (RENAR-2, self-assessment) — полная схема в §13.4.2:

```
renar-version: "1.1"
senar-version: "1.0"
manifest-version: 1
manifest-id: "CFM-2026-014"
level: "RENAR-2"
level-target: "RENAR-3"
assessment-mode: "self"
assessment-date: "2026-05-20"
assessor: { id: "architect-team-lead", role: "architect", signature-ref: "
<pointer>" }
next-assessment-due: "2026-08-20"
```

```
mandatory-clauses-confirmed:
  sot-inversion: true
  substrate-v1-v6: { v1: true, v2: true, v3: true, v4: true, v5: true, v6: true }
  adapt-per-tz: true
  spec-types-closed-list: true
  tc-pos-neg-pairing: true
  quality-gates-closed-list: true
  closed-lists-backward-findings: true
quality-gates: { qg-0: required, qg-1: required, qg-2: required, qg-3: absent,
qg-4: absent }
spec-types-supported: ["SPEC-ARCH","SPEC-API","SPEC-DATA","SPEC-INT","SPEC-
PROC","SPEC-UI","SPEC-AI","SPEC-SEC","SPEC-OPS","SPEC-TEST","SPEC-DOC","SPEC-UC"]
exceptions: []
replaced-by: null
```

11. Автогенерируемые compliance-артефакты

Генерируемые из существующих требований артефакты для аудиторов:

Artifact	Источник	Содержание
Records of Processing (GDPR Art.30)	BR/SR с `contains-pii: true`	Категории данных, lawful basis, retention, recipients
Statement of Applicability (ISO 27001)	BR/SR с `compliance: ISO 27001:2022`	Annex A controls в/вне scope, justification
Data Inventory	Все `data-classification` записи	Categories, residency, retention, owners
AI System Cards	SPEC-AI	Model card по NIST AI RMF / AI Act format
Отчёт журнала аудита	Traceability BR → SR → SPEC → TC → last-run	Цепочка от контроля до evidence
DPIA Index	`dpia/` папка	Список DPIA + статус согласования с DPO
AI Risk Register Snapshot	AI risk register	Все идентифицированные риски + mitigations + residual

Нативный для носителя reporting: агрегация доказательной базы из versioned artifacts (V4) в compliance report по запросу оценщика.

Evidence pack — шаблоны (informative; полный E2E — guide/09 §E3). GDPR Art.15 trace bundle — таблица `Control | BR/SR | SPEC | TC | last-run` + lawful basis + retention + ссылка на manifest. ФЗ-152 ст.14 — mapping table `Требование ↔ RENAR artifact ↔ Evidence field`. ISO 29148 trace excerpt — reference/07 (заполнить «RENAR frontmatter» для каждого SR в scope). Самооценка перед аудитом — reference/08 §2.

12. Что RENAR НЕ покрывает в compliance

RENAR — инфраструктура для compliance, но не **substitute** для: DPO (Data Protection Officer — человеческая роль, юридическая ответственность); legal review договоров и privacy notices; pen-testing реализации; bug bounty / vulnerability management; physical security (датацентры, офис); operational security (key rotation, incident response, monitoring runbook); cyber insurance; regulatory filings (уведомления РКН, GDPR registration с DPA, AI Act conformity assessment).

RENAR помогает делать compliance **в процессе разработки требований**. Operational compliance — отдельные процессы, которые *ссылаются* на RENAR-артефакты как на доказательную базу.

---

13. Resolved decisions для v1.0

- **Compliance frontmatter — conditional mandatory.** Default — optional; mandatory при `contains-pii: true` (GDPR/ФЗ-152 trigger) или `contains-phi: true` (HIPAA trigger). Артефакт с PII без compliance frontmatter — non-conformant (§13.3 расширения через manifest declared-stricter).
- **DPIA — mixed model.** Формальный DPIA — внешний документ (legal owns); RENAR-артефакт `dpia/<slug>.md` хранит machine-readable summary + pointer на formal document. Separation of concerns при traceability.
- **Data residency — вне scope RENAR.** Per §1.3 (3) tech stack/infra out of scope. Enforcement — DevOps-уровневые контролы (network policies, cloud regions). RENAR фиксирует **требование** (`data-classification.data-residency: ["EU"]`), не enforcement.
- **Custom industry frameworks** (ЦБ РФ финтех, HIPAA healthcare и др.) — отдельные документы (industry-specific addenda как `guide/06-compliance-<industry>.md` или внешние документы; **могут** declared-stricter поверх RENAR-conformance).

Отложено на v1.1 (бэклог фазы 8): автовыгрузка доказательств по Schrems II и трансферам ЕС ↔ РФ — частично закрывается флагами `cross-border-transfer` и ссылками на SCC, но полная автоматизация недостижима (какие SCC применимы — решают юристы). Ответственные: связка legal-tech / адаптеры.

---

14. Связь с другими главами

- 00-quickstart — базовый цикл RENAR без compliance-надстройки.
- 05-safe-comparison — RACI с SAFe ролями (включая DPO как Consulted).
- reference/02-schemas — frontmatter schemas: `compliance` , `data-classification` , `gdpr` , `ai-act` .
- reference/03-ai-risk-register — AI risk register структура.
- reference/04-ai-style-guide — стиль AI-провенанса.
- standard/04-terms — SPEC-SEC, SPEC-AI, SPEC-OPS терминология.
- standard/13-conformance — RENAR-уровни conformance (≠ compliance: RENAR-N оценивает зрелость *процесса*, compliance — соответствие *внешним нормам*).

---

07. Режимы отказа

> *Систематический обзор всех известных способов, которыми RENAR-проект может выйти из строя: технический дрейф между артефактами и реализацией, AI-специфичные риски, и (главное) организационные паттерны, при которых процесс существует на бумаге, но не работает. Классы дрейфа нормированы в standard/00 §0.3; AI-риски — в reference/03. Для каждого failure mode — симптом, как обнаружить, как предотвратить, как восстановиться.*
>
> ***Предпосылки:*** *RENAR Core, reference/03-ai-risk-register.md.*

1. Карта failure modes

Три класса проблем:

Класс	Где живёт	Как обнаруживается
Drift	Несоответствие между разными представлениями одной сущности (frontmatter ↔ DB, требование ↔ код, TC ↔ требование)	Reconciliation hook (drift detection, §4.11)
AI risks	Свойства AI-генерации (hallucination, bias, injection, model drift)	Adversarial review + eval-тесты + AI risk register (reference/03)
Organizational	Несоответствие между формальным процессом и реальными практиками команды	Поведенческие сигналы: паттерн утверждений, частота споров, частота обхода

Drift и AI risks ловятся механизмами носителя. Organizational — никаким носителем не ловятся; нужны human-уровневые рецензии процесса. Эта глава покрывает все три.

2. 8 классов дрейфа

Для каждого: симптом (как выглядит со стороны), detection (как обнаружить автоматически), prevention (как не допустить), recovery (что делать если уже случилось).

2.1 Schema drift

Симптом: Поля во frontmatter артефакта расходятся с теми, что носитель ожидает / поддерживает.

Обнаружение: На каждое изменение артефакта носитель валидирует frontmatter по schema (reference/02-schemas.md). При расхождении — блок integration (QG-0 фейлит).

Предотвращение: Schema — single source of truth (closed list), не редактируется на проекте. Изменения schema — только через изменение полного RENAR Standard.

Восстановление: Откатить frontmatter к schema-valid состоянию; если изменение нужно — открыть RFC на изменение стандарта.

2.2 Lifecycle drift

Симптом: Состояния (черновик / версия комплекта `N.M` / статусы TR и ADAPT) и названия контрольных точек качества понимаются по-разному в разных подсистемах или у разных команд.

Обнаружение: Сравнить переходы статусов в журнале аудита с нормативной state machine (standard/10-lifecycle-qg). Аномалии (переход без соответствующих pre-conditions) — флаг.

Предотвращение: Переходы выполняются механизмом носителя, не ручной правкой frontmatter. Capability V3 (state machine enforcement).

Восстановление: Откатить нелегитимный переход; провести QG-0 / QG-2 заново через корректный механизм.

2.3 Source-of-truth drift

Симптом: Одна и та же сущность редактируется в двух местах (например, и в `.req` директории, и в Jira-трекере). Версии расходятся.

Обнаружение: Periodic reconciliation между носителем и tracker; diff показывает расхождения.

Предотвращение: В каждый момент времени для проекта **выбран ровно один SSoT-носитель**. Tracker — derived view, не источник истины. Hook носителя блокирует tracker-only изменения требований.

Восстановление: Объявить один носитель-winner; смерджить второй в первый; убрать редактирование во второй до миграции.

2.4 Implementation drift

Симптом: Код в реализации ссылается на SR, которого больше нет (deprecated, удалён, переименован). Или: SR существует, но реализация ушла от него (поведение не соответствует).

Обнаружение: Reconciliation hook (drift detection):

- Forward: пройти по requirement → найти реализующий код → запустить TC.
- Backward: пройти по коду → найти ссылки на SR / TC → проверить, что они входят в верифицированную версию комплекта.

Предотвращение: ID требований **неизменяемы** — переименование запрещено. Deprecated требования остаются в репозитории со статусом `deprecated`, не удаляются.

Восстановление: Открыть delta-T3, который явно adopts текущую реализацию (или, наоборот, требует откат кода до соответствия требованию).

2.5 Terminological drift

Симптом: «Verified», «implemented», «approved» означают разное у разных людей / команд.

Обнаружение: Code review checklist: «использован термин не из глоссария?» — флаг. Аналогично — валидатор носителя проверяет, что значения enum-полей frontmatter только из closed list.

Предотвращение: Глоссарий — единственный источник терминов (reference/01-glossary). Каждый термин = ровно одно состояние lifecycle.

Восстановление: Провести ревизию всех артефактов проекта на использование out-of-glossary терминов; заменить или подать RFC на расширение глоссария.

2.6 Order / provenance drift

Симптом: Delta-T3 #2 ссылается на SR, который был создан в Delta-T3 #1, но применение пошло в обратном порядке — SR не существовал на момент применения #2.

Обнаружение: Delta-T3 нумеруются и применяются строго в порядке номеров. Hook носителя проверяет, что upstream delta уже применена.

Предотвращение: Delta-T3 нельзя перенумеровать. Каждый артефакт хранит `created-by-order` (delta-T3 создания) и `last-modified-by-order` (последний апдейт).

Восстановление: Откатить out-of-order применение; перепримерить в правильном порядке.

2.7 TC ↔ requirement provenance drift

Симптом: TC верифицирует требование, но норма уже изменена в новой версии комплекта — `automation.set-version` реализации ниже версии, в которой норма изменена. Тест зелёный, но проверяет устаревшее поведение.

Обнаружение: Coverage report показывает категорию «Stale» — реализации с устаревшим `automation.set-version` (standard/10 §10.9.4). Reconciliation ловит это автоматически (по V5-pin версии комплекта).

Предотвращение: В TC обязательное поле `automation.set-version` — версия комплекта, против которой написана реализация. QG-2 не записывает верификацию версии, если хотя бы одна реализация устарела или её `last-run.set-version` не равен версии.

Восстановление: Перепрогнать stale TC на текущей версии требования; обновить, если TC сам устарел.

2.8 Test-fitting drift

Симптом: AI-агент имеет тривиальный путь зеленения failing-теста — ослабить Pass/Fail-критерий вместо исправления кода. Без защиты тесты дрейфуют от «строгий проверщик» к «зелёная пустота».

Обнаружение: Изменение Pass/Fail-критериев в TC без явного `[test-spec-change]` тега — флаг носителя. Periodic spot-check 5 случайных passing TC раз в спринт.

Предотвращение:

- MR / change, изменяющий Pass/Fail-критерии, обязан иметь тег `[test-spec-change]` и отдельный approval инженера (не совмещённый с approval кода-фикса).
- Изоляция judge-модели: production-модель ≠ judge-модель.
- Метрика test-fitting drift rate trending.

Восстановление: Восстановить старые критерии; провести root cause analysis — почему AI-агент выбрал зеленение вместо фикса; обновить prompt / system instructions.

---

3. 14 AI-рисков (краткая сводка)

Полные описания, mitigations и owner — в reference/03-ai-risk-register. Здесь — operational summary: id, название, severity, главный detection signal.

ID	Название	Severity	Detection signal

AIR-01	Hallucination в AI-генерируемых требованиях	High	Hallucination Rate metric > threshold; adversarial critic flags
AIR-02	Prompt injection через ТЗ от клиента	High	Suspicious pattern в imports; sandbox violation
AIR-03	Model drift / version change	Medium	diff regression при смене модели; сбой эталона eval
AIR-04	Bias в AI-генерации требований	Medium	Stakeholder map gaps; missing accessibility/locale considerations
AIR-05	Single-model failure (no diversity)	Medium	Все артефакты с одним `ai-provenance.model`; нет multi-model agreement
AIR-06	Test-fitting / зеленение тестов	High	diff в TC Pass/Fail без `[test-spec-change]` тега
AIR-07	Hallucinated citations	Medium-High	Citation validator hook fails
AIR-08	Adversarial inputs в client data	High	Application-level (out-of-scope RENAR, tracked в SPEC-SEC)
AIR-09	Privacy leakage через AI logs	High	PII в tool_event audit; redaction скип
AIR-10	Knowledge graph poisoning	Medium	Incorrect edges; circular dependencies в graph
AIR-11	Reconciliation false positive overload	Low-Medium	Findings/week trending up без real issues; dismissal rate высокий
AIR-12	Cost runaway (uncontrolled AI spend)	Medium	Project AI cost approaching budget cap
AIR-13	Stakeholder не понимает AI-сгенерированные требования	Medium	Dispute rate at acceptance растёт; длинные циклы утверждения
AIR-14	Vendor lock-in to specific LLM provider	Medium	All prompts работают только на одном provider

Risk matrix и периодичность рецензирования — reference/03 §5-§2.

3.5 Adversarial review (процедура)

> ***Информативно.*** *Операционная процедура для WC-13; нормативные требования — standard/09 §9.4, standard/13 §13.2 (RENAR-5).*

Когда обязательно (normative): adversarial review — QG-0 для RENAR-5 (§11.8.1); для `SPEC-SEC` / `SPEC-AI` — external reviewer на QG-0 (§5); declared-stricter может расширить scope (standard/00 §0.6).

Шаг	Актор	Артефакт	Exit criterion

1. Scope	Architect	Список TC + связанные SR/SPEC	Каждая TC-норма утверждённой версии в scope имеет `tc-type` и `verifies[]`
2. Critic pass	AI-критик (отдельная модель/ промпт)	Журнал находок с id, severity, ссылкой на TC/SR	Находки прослеживаемы к конкретному clause §9.x; без «общих» рекомендаций
3. Triage	Architect + RE engineer	Disposition: fix / accept / reject	Каждый finding — owner + rationale; dismiss без rationale запрещён (см. §4.6)
4. Re-run	AI-агент или human	Обновлённые TC + diff	QG-2 pre-condition: `passing-tests / total-tests` для scope (§9.10)
5. Журнал аудита	носитель (V1)	commit/change unit с `adversarial-review` tag	provenance: model id, prompt version, findings hash (§10.13)

Дисциплина утверждений: метрики «100%» в §9 — это **target при QG-2**, не гарантия качества продукта. Severity AI-рисков — из reference/03, не редакторское переопределение.

Agent panel (без human reviewers): informative процедура — §3.5 (шаги 1–5); rubric и severity — reference/03.

---

4. Organizational failure patterns

Эти проблемы не ловятся механизмами носителя — это поведенческие паттерны команд. Появляются типично через 2-6 месяцев после внедрения RENAR.

4.1 ACTZ как формальность

Симптом: Протокол уточнения T3 подписывается не глядя. Вопросы вынесены клиенту сырым выводом агента — списком на три страницы, из которого клиент не понимает, за что расписывается. Backward-секция ADAPT пустая или содержит yes/no без контекста.

Признак: ACTZ подписан за < 24 часов после генерации черновика; доля спорных требований на приёмке растёт.

Смягчение: Клиенту выносится **подготовленный** список решений, а не сырой вывод агента (standard/07 §7.13.4): вопросы агрегирует и переформулирует архитектор, на языке обязательств. Двусторонняя подпись — у ACTZ, и только у него; ADAPT подписывает один архитектор (§7.5), и требовать под ним подпись клиента бессмысленно — это и была та фикция, которая порождала «подписал не читая». Backward-секция обязана содержать ≥ 1 не риторический вопрос. Spot-check ACTZ и ADAPT в I&A.

4.2 SPEC overload

Симптом: Команда создаёт SPEC для каждой задачи, даже когда SR + TR достаточно. SPEC-каталог разрастается; каждый PR обновляет 5+ SPEC.

Признак: Rate SPEC / SR > 1.5 (ожидаемое < 0.3 для проектов средней сложности).

Смягчение: Pre-review checklist: «нужен ли SPEC для этого изменения?» SPEC оправдан только когда есть несколько SR с общим constraint. См. standard/08-specifications.md §8.2 — когда SPEC

обязателен.

4.3 Hooks как препятствие

Симптом: Команда регулярно обходит hooks носителя (`--no-verify` , манипуляции с timestamp, ручная правка statuses).

Признак: Git log / журнал аудита носителя показывает частоту обхода commits; QG-0/QG-2 проходят за подозрительно короткое время.

Смягчение: Root cause — hooks слишком медленные / слишком noisy / слишком жёсткие. Не «запретить обход», а починить hooks. Метрика частоты обхода как trending — если растёт, провести retro с командой.

4.4 Drift detection без действия

Симптом: reconciliation hook генерирует находки дрейфа, но никто на них не реагирует. Бэклог находок растёт, старые находки игнорируются.

Признак: Находки старше 14 дней > 30; resolution rate < 20% / неделю.

Смягчение: Каждая находка дрейфа — owner и SLA (resolve / accept / reject в течение N дней). Неразрешённые находки выше SLA — escalation. Reconciliation без human ownership = шум.

4.5 tracker as parallel universe

Симптом: Команда живёт в Jira / Linear / ADO; `.req` директория обновляется раз в неделю «для галочки». Tracker — реальный источник истины, RENAR — формальный артефакт для аудита.

Признак: diff `.req` vs tracker > 30% по any given week; commits в `.req` редкие и батчевые.

Смягчение: Single source of truth должен быть размещён в носителе, не tracker-resident. Tracker — derived view только. Если команда не может работать без tracker — носитель должен пушить *в* tracker, не наоборот.

4.6 Critic burnout

Симптом: AI-критик (adversarial review) генерирует много находок; постепенно дев / архитектор начинают игнорировать его output. Находки отклоняются без рассмотрения.

Признак: Dismissal rate AI-критика > 80%; time-to-dismiss < 30 секунд per finding.

Смягчение: Tunable thresholds для критика. Если ratio false positive высокий — recalibrate prompt / model. Метрика «находки критика → real issue» (% от отклонённых находок, которые потом всплыли как defect) — если 0%, критик useless.

4.7 Single-engineer dependence

Симптом: Только один инженер на проекте «понимает RENAR». Все QG-0 / QG-2 проходят через него. Если он в отпуске — процесс встаёт.

Признак: Bus factor RENAR-владения = 1. Distribution of QG approvals heavily skewed к одному человеку.

Смягчение: Парный onboarding (минимум 2 инженера на проекте умеют RENAR). Rotation QG-approver роли. Documentation проектных конвенций в `<project>.req/CONVENTIONS.md` .

4.8 Ad-hoc delta

Симптом: Изменения требований происходят без оформления delta-T3 — «давай просто поменяем SR-12 прямо в репозитории».

Признак: Direct commits в `<system>.req/sr/*` без соответствующего delta-T3; `created-by-order` поле пустое.

Смягчение: hook носителя блокирует mutation существующих требований без `delta-ref` в commit metadata. Все изменения — через delta-T3 workflow (standard/07-adapt §7.6).

4.9 TC abandonment

Симптом: TC создаются вместе с требованиями, но затем не прогоняются. `last-run` старше N месяцев; coverage report показывает «зелёные» TC, которые в реальности никогда не запускались за полгода.

Признак: Median `last-run` age > 90 дней; TC count растёт, run count не растёт.

Смягчение: носитель автоматически прогоняет TC по schedule (capability V4). Реализации без `last-run` за N дней автоматически помечаются устаревшими; QG-2 блокирует запись верификации, пока не перепрогнаны.

4.10 Тест, рождённый зелёным

Симптом: TC написан тем же агентом и в той же сессии, что и реализация, — и с рождения зелёный. Формально покрытие есть; фактически тест проверяет код, а не требование.

Признак: В истории прогонов TC нет ни одного красного результата; провенанс TC совпадает с провенансом единицы изменения реализации.

Смягчение: изоляция авторства по трём осям — время (TC-норма утверждена в версии и реализация допущена до старта TR), автор (агент теста ≠ агент реализации) и изменение (правка Pass/Fail только через `[test-spec-change]`), standard/09 §9.18. Фиксирующий прогон до реализации обязан быть красным; зелёный на нём — сигнал разбора, а не успех. Единственное исключение — TC класса `implementation-originated`, у которого красной истории не бывает по построению: там зубастость доказывается убитым мутантом.

4.11 Приёмка, подтверждающая саму себя

Симптом: AT сгенерированы агентом, который «для контекста» получил доступ к SR и ADAPT — или просто той же моделью, что писала реализацию. Все AT зелёные, приёмка у заказчика проваливается.

Признак: `generator` у AT не подтверждает изоляцию; либо `tz-version` у AT отстаёт от действующей редакции итогового ТЗ.

Смягчение: AT выводятся **только** из итогового ТЗ, изолированным агентом на другой модели, без доступа к внутреннему контуру (standard/09 §9.19.2) — иначе уровень приёмки схлопывается в уровень верификации и ошибка толкования получает ложное подтверждение соответствия. AT перегенерируются перед **каждыми** испытаниями: программа, выведенная при планировании, к концу длинного заказа проверяет систему против контракта годичной давности.

---

5. Failure recovery playbook

Что делать, когда система уже сломана. Последовательность общая для всех failure modes; specifics зависят от класса.

Шаг 1: Stop the bleeding

Найти и остановить ongoing damage:

- Drift: заморозить дальнейшие изменения в затронутой области.
- AI risk: приостановить AI-генерацию для затронутого класса артефактов.
- Organizational: вынести на retro / I&A — это не technical fix.

Шаг 2: Quantify

Измерить ущерб:

- Сколько артефактов в drift-состоянии?
- Сколько релизов с момента возникновения проблемы?
- Какие SR / SPEC / TC затронуты? (Capability V4 — coverage / drift report)

Шаг 3: Triage

Сегментировать ущерб на:

- **Critical** — уже в production, влияет на пользователей. Hot-fix.
- **Active** — в текущем PI, влияет на ongoing work. Block PI exit.
- **Historical** — старые артефакты, не активно используются. Batch fix.

Шаг 4: Fix

Для каждого класса — соответствующий fix:

- Schema drift → откат frontmatter; RFC если schema нужно расширить.
- Implementation drift → delta-T3 adopt OR откат кода.
- TC drift → пересоздать реализацию против текущей версии комплекта (`automation.set-version`).
- Test-fitting → revert критерии; root cause AI-агента.
- Organizational → process retro + специфичные mitigations (§4).

Шаг 5: Prevent recurrence

- Усилить detection (нижний threshold, новая metric).
- Добавить mitigation в processed артефакт.
- Зафиксировать lessons learned в project decision log или ADAPT backward findings (категория `scope` / `terminology`).

Шаг 6: Verify

После fix — пройти QG-2 на затронутых артефактах заново. Drift detection должна показать clean state.

6. Negative: чего эта глава не покрывает

- **Security incidents** — breach response, forensics, regulatory notification. Это процесс уровня organization security, не RENAR scope.
- **AI red team / penetration testing** — отдельный security workflow; RENAR трекает только что соответствующие SR / SPEC-SEC должны быть.
- **Compliance breach response** — нарушение GDPR / ФЗ-152 / PCI-DSS требует юридического процесса с DPO / regulator, не technical recovery.
- **Production incidents** — outages, performance regressions. Это operational, см. SPEC-OPS runbook.
- **Stakeholder conflicts** — диспуты на acceptance, scope disagreements. RENAR даёт журнал аудита (кто что approved когда), но resolution — human process.

7. Связь с другими материалами о режимах отказа

Документ	Что в нём	Когда читать
reference/03-ai-risk-register	Полный реестр 14 AIR-рисков с mitigations	При планировании AI use-case; при review eval-стратегии
standard/04-terms §4.11	Closed list drift классов с нормативными определениями	При спорах о терминологии failure modes
05-safe-comparison §9	RACI matrix — кто accountable за каждую активность	При расследовании organizational failure
reference/04-ai-style-guide	Стиль AI-провенанса; минимальный контракт для AI-сгенерированных артефактов	При диагностике AIR-01 (hallucination), AIR-07 (citations)

8. Resolved decisions для v1.0

- **Набор шагов восстановления без привязки к платформе.** Последовательность из §5 носит универсальный характер. Детали «как именно заморозить изменения» для `git` и document-store — в 03-tool-guide-git §3 и 04-document-store-substrate. Объём нормативного минимума задан здесь же, в главе 7.
- **Подстройка критика событийно.** Повторная настройка промпта критика выполняется при выходе за порог метрик drift / галлюцинаций (§12.3.3); уровень RENAR-5 требует непрерывной оценки (§11.8.1), поэтому регулярный «общий пересмотр без причины» избыточен. По срабатыванию метрик — допустимо.

8.1 Отложено на v1.1 (бэклог фазы 8)

- **Числовые пороги для организационных паттернов (§4).** Сейчас даны качественные «признаки». Набор допустимых значений понадобится после накопления полевых данных.

Ответственные: команда стандарта RENAR и внедренческие организации.

- **Формальный замер коэффициента «техавтобусности» для §4.7.** Вспомогательные средства не зафиксированы; возможный подход — графовый запрос по авторам коммитов в цепочке ревизий (встроенное в носитель сочетание **V6**). Ответственные: авторы средств для конкретных сред хранения.
- **Калибровка provisional-порогов метрик для RENAR-4 / RENAR-5.** Целевые значения §12.4 заданы как ориентиры направления и подлежат уточнению, когда по метрике накопятся данные от пяти независимых организаций при завершённом цикле у каждой; до уточнения действуют как есть, но превышение неоткалиброванного порога само по себе соответствие не снимает. Условие выполненным объявляет только процедура изменения стандарта (§13.9) публикацией отчёта о калибровке — не отдельное лицо и не внедренческая организация.

---

08. Руководство разработчика

> ***Пример только для среды*** `git`***.*** *Эта глава иллюстрирует* ***один*** *возможный сценарий на примере GitHub и разнесённых по репозиториям подсистем (каталоги* `.req` *и* `.src` *).* ***RENAR к конкретной реализации среды не привязан****; ваши перехватчики и пайплайны могут быть устроены иначе. Общезначимые главы — §6 Иерархия требований, §8 Спецификации, §10 Жизненный цикл и контрольные точки качества. Альтернатива без VCS как файловой базы — 04-document-store-substrate.md.*
>
> ***Выдуманная компания AcmeCorp:*** *платформа* `acme-platform` *и четыре подсистемы —* `acme-portal` *,* `acme-ai` *,* `acme-orchestrator` *,* `acme-site` *. Имена в примерах замените на свои; ссылки на нормативные главы этого не затрагивают.*

1. Что нужно знать перед стартом

Два типа репозиториев. Каждая подсистема имеет два отдельных репозитория:

Репозиторий	Суффикс	Что внутри	Кто пишет
Требования	`.req`	BR, SR — что система должна делать	Архитектор, Tech Lead
Исходный код	`.src`	Код, тесты, CI/CD	Разработчик

Разделение намеренное: требования версионируются независимо от кода, имеют другой цикл рецензирования и другие права доступа.

Иерархия: Система (`acme-platform`) → Подсистема (`acme-portal` , `acme-ai` , `acme-orchestrator` , `acme-site`) → Модуль (если есть). Каждый уровень имеет свой `.req` репозиторий; требования верхнего уровня — родители для требований ниже.

Три уровня требований:

```
BR  — Бизнес-требование  (зачем это нужно бизнесу)
 └── SR  — Системное требование  (что делает система)
      └── TR  — Требование к задаче  (конкретика для реализации)
                ↑ это поля вашей задачи в трекере, не файл
```

TR не является файлом — это Goal + Acceptance Criteria в задаче трекера.

2. Первоначальная настройка локального окружения

Шаг 1. Уточните у Tech Lead, с какой подсистемой работаете: `acme-portal` (клиентский портал), `acme-ai` (AI-pipeline), `acme-orchestrator` (оркестратор), `acme-site` (публичный сайт).

Шаг 2-3. Создайте структуру и клонируйте репозитории:

```
mkdir -p ~/projects/acmecorp/acme-platform && cd ~/projects/acmecorp/acme-
platform

# Обязательно для всех — родительские требования системы (только чтение):
git clone git@github.com:acmecorp/acme-platform/acme-platform.req

# Обязательно — репозитории вашей подсистемы:
SUBSYSTEM=acme-portal   # замените на свою
mkdir -p $SUBSYSTEM
git clone git@github.com:acmecorp/acme-platform/$SUBSYSTEM/$SUBSYSTEM.req
$SUBSYSTEM/$SUBSYSTEM.req
git clone git@github.com:acmecorp/acme-platform/$SUBSYSTEM/$SUBSYSTEM.src
$SUBSYSTEM/$SUBSYSTEM.src

# По необходимости — требования смежных подсистем (только чтение):
mkdir -p acme-ai && git clone git@github.com:acmecorp/acme-platform/acme-ai/acme-
ai.req acme-ai/acme-ai.req
```

Шаг 4. Проверка: `find ~/projects/acmecorp -maxdepth 4 -name ".git" -type d | sed 's|/.git||' | sort`. Ожидаемый результат для разработчика Acme Portal:

```
~/projects/acmecorp/acme-platform/acme-platform.req
~/projects/acmecorp/acme-platform/acme-portal/acme-portal.req
~/projects/acmecorp/acme-platform/acme-portal/acme-portal.src
```

---

3. Структура папок на локальной машине

Локальная структура **зеркалит** иерархию репозиториев в хостинге — относительные пути `../..` между репозиториями становятся предсказуемыми.

> *С раскладкой по каталогам (раскладка носителя) см. guide/03 §2 и §4: каноническая схема репозитория + закрепление `.src` → `.req` через submodule (возможность V5). Здесь — типичное локальное размещение для IDE: несколько рядом лежащих клонов без обязательного submodule в рабочей папке.*

```
~/projects/acmecorp/acme-platform/
  acme-platform.req/          # требования системы (read-only)
  acme-portal/
    acme-portal.req/          # требования вашей подсистемы
    acme-portal.src/          # код вашей подсистемы — здесь вы работаете
  acme-ai/{acme-ai.req,acme-ai.src}/             # клонируется по необходимости
  acme-orchestrator/{acme-orchestrator.req,acme-orchestrator.src}/
  acme-site/{acme-site.req,acme-site.src}/
```

> ***Правило:*** *не меняйте имена папок — они совпадают с именами проектов в git-хостинге. Это позволяет скриптам и CI работать с одними путями везде.*

4. Настройка рабочего пространства в IDE

VS Code Workspace. Создайте `<subsystem>.code-workspace` в папке подсистемы:

```
cat > ~/projects/acmecorp/acme-platform/acme-portal/acme-portal.code-workspace <<
'EOF'
{
  "folders": [
    { "path": "acme-portal.src", "name": "Code — Acme Portal" },
    { "path": "acme-portal.req", "name": "Requirements — Acme Portal" },
    { "path": "../acme-platform.req", "name": "Requirements — System (read only)"
}
  ],
  "settings": { "files.exclude": { "**/.git": true } }
}
EOF
```

Открыть: `code ~/projects/acmecorp/acme-platform/acme-portal/acme-portal.code-workspace`. В боковой панели — три репозитория одновременно.

JetBrains (IntelliJ, WebStorm, PyCharm). Откройте каждый репозиторий как отдельный модуль через **File** → **Attach Project** или **Project Structure** → **Modules**.

5. Работа с требованиями

5.1 Как читать. Перед началом задачи прочитайте SR (`acme-portal.req/sr/SR-NN-*.md`). В frontmatter SR найдите поле `parent` — ссылка на родительский BR:

```
parent: { id: BR-01, repo: "acmecorp/acme-platform/acme-platform.req", file:
"br/BR-01-order-ai-dev.md" }
```

Откройте родительский BR — это «зачем существует функциональность».

5.2 Трассировка. Цепочка: `Задача TASK-42 → SR-01 (acme-portal.req/sr/...) → BR-01 (acme-platform.req/br/...)`. Если что-то непонятно — поднимайтесь вверх.

5.3 Изменить требование. Разработчик создаёт Issue в `.req` репозитории с описанием несоответствия SR ↔ реальное поведение. Tech Lead / Архитектор вносит изменение:

```
cd acme-portal.req
git checkout -b change/TZ-2026-002-totp
# Редактируете файл SR; обновляете version + updated в frontmatter; обновляете
```

```
source.tz-section на раздел delta-T3
git add sr/SR-01-auth.md
git commit -m "[delta:TZ-2026-002] SR-01 v1.2: добавить TOTP"
# Создаёте MR/PR в git-хостинге
```

> ***Запрещено:*** *коммитить изменения требований напрямую в* `main` *без MR/PR и ревью.*

5.4 Что нельзя делать. Менять `acme-platform.req` без согласования с архитектором; менять требования смежных подсистем (`acme-ai.req` , `acme-orchestrator.req`); удалять файлы требований (только переводить в `status: deprecated`); создавать файлы версий (`SR-01-v1.md` , `SR-01-v2.md`) — история в `git log` .

---

6. Работа с задачами

6.1 Перед тем как взять задачу — все пункты QG-0 Approval Gate выполнены (reference/01 §2.4; pre-v1.0 legacy «Context Gate»):

- Сформулирована цель (Goal); есть Acceptance Criteria — конкретные, тестируемые, независимые.
- Есть хотя бы один негативный сценарий в AC.
- Задача ссылается на SR (поле в трекере); назначен тип работы.
- Если затрагивает безопасность — `threat-surface` задекларирован.

Если чего-то нет — **не берите задачу в работу**, вернитесь к Супервизору / Tech Lead.

6.2 Acceptance Criteria. Каждый AC — отдельный тест. Хорошие AC: `POST /auth/login возвращает 200 и JWT-токен при верных credentials` ; `POST /auth/login возвращает 401 при неверном пароле` (негативный сценарий); `POST /auth/login возвращает 422, если поле email отсутствует` ; `JWT-токен истекает через 24 часа` .

Плохие AC (задачу брать нельзя): «Логин должен работать корректно»; «Обработка ошибок должна быть надёжной»; «Система должна быть безопасной».

6.3 Если AC непонятен или противоречит SR: прочитайте SR + родительский BR; создайте комментарий в задаче с конкретным вопросом; не интерпретируйте молча — AI интерпретирует молча, именно поэтому и нужны чёткие требования.

---

7. Git-процесс

7.1 Работа с кодом (`.src`). Стандартный feature-branch workflow:

```
cd acme-portal/acme-portal.src
git checkout main && git pull
git checkout -b feat/TASK-42-totp-auth
# ... работа ...
git add src/auth/totp.py
```

```
git commit -m "feat(auth): реализовать TOTP-аутентификацию (TASK-42)"
# Создать MR/PR в git-хостинге
```

Формат коммита: `<type>(<scope>): <описание> (<TASK-ID>)`.

7.2 Работа с требованиями (`.req`). Ветка для изменения требования (см. §5.3 выше).

7.3 Привязка MR/PR к задаче. В описании MR/PR указывайте: `Closes #42` + `Related SR: SR-01 (acme-portal.req)`. Если изменение затрагивает несколько `.req` репозиториев — `Related: acmecorp/acme-platform/acme-platform.req#15`.

7.4 Именование веток:

Что делаете	Ветка
Новая функциональность	`feat/TASK-NN-slug`
Исправление бага	`fix/TASK-NN-slug`
Изменение требования	`change/TZ-YYYY-NNN-slug`
Новое требование	`feat/BR-NN-slug` или `feat/SR-NN-slug`

---

8. Частые сценарии

Сценарий 1. Новая задача: открыть в трекере → проверить QG-0 → найти ссылку на SR → открыть SR в `acme-portal.req/sr/` → прочитать SR + родительский BR (если нужен контекст) → прогнать привязанные к задаче TC (они уже заморожены и **красные** — это норма: их писали до вас и не вы, standard/09 §9.18) → создать ветку `feat/TASK-NN-slug` в `.src` → реализовать, пока TC не позеленеют → MR/PR → ревью → merge.

> ***Почему тест уже написан, а не пишется вами.*** *Тест, рождённый рядом с кодом, проверяет код, а не требование. Поэтому TC замораживается до старта задачи и пишется другим агентом или инженером. Если по ходу реализации вы добавили внутреннюю техническую деталь, которой нет в требованиях (защитная проверка, журналирование, обработка граничного случая), — её можно легализовать классом `source: implementation-originated` (standard/06 §6.13): нужны провенанс, явное одобрение Супервизора-человека, покрывающий тест до слияния и **убитый мутант** (у такого теста нет красной истории — он рождается зелёным). Наблюдаемое клиентом поведение этим классом не легализуется: экран, поле, сообщение, срок — только через контрактный контур.*

Сценарий 2. Несоответствие между SR и требованием задачи: НЕ делать ничего молча → комментарий в задаче («SR-01 §4 описывает X, задача требует Y — противоречие, прошу уточнить») → дождаться ответа Супервизора / Архитектора → не брать задачу в работу до устранения.

Сценарий 3. Понять откуда взялось требование. В `.req` репозитории: `git log -- sr/SR-01-auth.md` (история изменений) или `git show <commit-hash>` (детали конкретного изменения). Или в frontmatter файла найдите поле `source` — ссылка на ТЗ и его раздел.

Сценарий 4. Интеграция (Acme Portal → Acme AI). Клонируйте требования смежной подсистемы, добавьте в workspace; откройте `acme-platform.req/specs/int/SPEC-INT-01-acme-portal-acme-ai.md` — там описан контракт. Интеграционные контракты canonical-формы — `SPEC-INT-NN` (standard/08 §8.5.4), не legacy `INT-SR`. SR подсистемы ссылается через `constrained-by: [SPEC-INT-NN]`. SPEC-INT всегда живёт в `acme-platform.req/specs/int/` — не внутри подсистем.

Сценарий 5. Дельта-ТЗ. Работа архитектора/Tech Lead, но разработчику: дождаться merge MR с обновлёнными SR → `git pull` в `.req` → прочитать `tz/TZ-YYYY-NNN-delta-N.md` (текст delta-ТЗ) → проверить, затрагивают ли изменения ваши текущие задачи → если да — обновить или закрыть по согласованию с Супервизором.

Сценарий 6. Обновить локальные репозитории требований:

```
find ~/projects/acmecorp -name "*.req" -type d | while read repo; do
  echo "Updating $repo..."
  git -C "$repo" pull --ff-only
done
```

9. Права доступа — кто что может менять

Репозиторий	Разработчик	Tech Lead	Архитектор
`acme-platform.req` (системные BR/SR)	Только чтение	Только чтение	Запись через MR
`acme-portal.req` (SR подсистемы)	Только чтение	Запись через MR	Запись через MR
`acme-portal.src` (код)	Запись через MR	Запись + ревью	—
`acme-ai.req` (чужая подсистема)	Только чтение	Только чтение	—

Если нужно изменить требование — создайте Issue в `.req` репозитории, не коммитьте напрямую.

10. Быстрый справочник

Куда смотреть когда непонятно:

Вопрос	Где
Что должна делать система?	`acme-portal.req/sr/SR-NN-*.md`
Зачем это нужно бизнесу?	`acme-platform.req/br/BR-NN-*.md`
Откуда взялось это требование?	frontmatter `source` → ТЗ
Как изменилось требование?	`git log -- sr/SR-NN-*.md`

Как работает интеграция с Acme AI?	`acme-platform.req/specs/int/SPEC-INT-01-*.md`
Что изменилось по последнему ТЗ?	`acme-platform.req/tz/TZ-YYYY-NNN-delta-N.md`

Чеклист перед созданием MR: код покрывает все AC из задачи; есть тесты на негативные сценарии; MR/PR содержит ссылку `Closes #NN`; если изменилось поведение — создан Issue в `.req` для обновления SR.

Команды Git (наиболее частые):

```
git -C <portal.req> pull                                    # обновить требования
git -C <portal.req> log -- sr/SR-01-auth.md                 # история конкретного SR
grep -r "id: SR-01" ~/projects/acmecorp/ --include="*.md"   # найти требование по ID
grep -r "SR-01" ~/projects/acmecorp/ --include="*.md"       # все задачи, ссылающиеся на SR-01
```

Глоссарий: BR — бизнес-требование; SR — системное требование; TR — Goal + AC в трекере; AC — Acceptance Criteria; QG-0 — Approval Gate (canonical v1.0; pre-v1.0 legacy «Context Gate»); `.req` — git-репозиторий с требованиями подсистемы; `.src` — git-репозиторий с кодом; SPEC-INT — интеграционная спецификация (canonical v1.0; pre-v1.0 `INT-SR`); delta-ТЗ — дополнение к ТЗ при повторном заказе; deprecated — устаревшее требование (не удаляется, только помечается).

09. Библиотека примеров

> *Восемь сценариев (E1–E8): шесть примеров in-doc (E3–E8) + два walkthrough (E1–E2). Каждый in-doc пример проходит цепочку **T3** → **ADAPT** → **BR/SR** → **SPEC** → **TC** → **QG** целиком или частично.*

#	Сценарий	Документ	Аудитория	Акцент
E1	Email/password sign-up (минимальный)	00-quickstart	Новичок	30 мин, минимум артефактов
E2	Login + profile + permissions (полный)	01-walkthrough	Tech Lead, QA	Полный lifecycle, AI-генерация TC
E3	Экспорт персональных данных (GDPR / ФЗ-152)	§2	Legal, PM, Architect	Compliance, SPEC-DATA/SEC
E4	Webhook idempotency (REST API)	§3	Backend, Architect	`tc-type: contract`, SPEC-API
E5	RAG-ассистент (SPEC-AI eval)	§4	AI engineer	`tc-type: eval`, judge isolation
E6	Delta-T3: scope dispute	§5	PM, Client, Architect	ADAPT backward, неизменяемое T3
E7	Подсистема как самостоятельный продукт	§6	Architect	Ребро `implements[]`, §6.8.2
E8	Интерфейсная часть E3: пример до конца	§7	UI/UX, Architect, QA	Заполненные тела SR / SPEC-UI / `ux` -TC

2. E3 — Экспорт персональных данных (GDPR Art. 15 / ФЗ-152)

Контекст: SaaS «AcmeCRM» хранит PII клиентов. Регулятор и договор обязывают выдать машиночитаемый экспорт по запросу субъекта данных за ≤30 дней.

2.1 Фрагмент T3 (immutable)

```
# TZ-2026-002 — Data subject export

§3.1 Субъект данных может запросить полный экспорт своих PII через UI «Privacy → Export my data».
§4.2 Формат: JSON + CSV bundle, zip, checksum SHA-256.
§4.3 SLA: готовность ссылки на скачивание ≤ 72 часа с момента verified identity.
§4.4 Экспорт включает: profile, activity log 24 мес., marketing consents.
§4.5 Запрос логируется; повторный экспорт — не чаще 1 раза в 30 дней без переопределения администратором.
```

2.2 ADAPT (сокращённо)

```yaml
id: ADAPT-002
type: ADAPT
status: approved
source-tz: { id: TZ-2026-002, signed-date: "2026-05-01" }
```

Forward §3: экспорт асинхронный (job queue); identity — через существующий MFA flow.

Backward (resolved):

#	Finding	Resolution
B-01	«24 мес. activity» — calendar или rolling?	Rolling 730 days
B-02	Override admin — кто approves?	Role `privacy-officer` + журнал аудита

2.3 BR + SR

```yaml
# br/BR-02-data-subject-export.md
id: BR-02
type: BR
title: "Субъект данных получает машиночитаемый экспорт PII"
source: { adapt: ADAPT-002, adapt-section: "Forward §3" }
compliance:
  - { standard: GDPR, control: "Art. 15" }
  - { standard: "ФЗ-152", control: "ст.14" }
```

```yaml
# sr/SR-08-export-request.md
id: SR-08
type: SR
parent: { id: BR-02 }
title: "Authenticated user инициирует export job"
constrained-by: ["SPEC-API-04", "SPEC-DATA-02", "SPEC-SEC-03"]
```

```yaml
# sr/SR-09-export-delivery.md
id: SR-09
type: SR
parent: { id: BR-02 }
title: "User скачивает готовый bundle по time-limited signed URL"
constrained-by: ["SPEC-API-04", "SPEC-SEC-03"]
```

2.4 SPEC (выдержки)

SPEC-DATA-02 — schema export bundle (tables: `users`, `activity_events`, `marketing_consents`).

SPEC-SEC-03 — signed URL TTL 24h; rate limit 1 export / 30 days; role `privacy-officer` для override.

SPEC-API-04 — `POST /v1/privacy/export`, `GET /v1/privacy/export/{job_id}`.

2.5 TC (pos/neg для SR-08)

```
---
id: TC-080
title: "Export job создаётся для verified user"
type: TC
tc-type: system
automation: { status: automated, set-version: "1.0" }
verifies:
  - id: SR-08
negative: false
---
## When
POST /v1/privacy/export (session: verified-user)
## Then
- 202 + job_id; status pending; audit event recorded
```

```
---
id: TC-081
title: "Export отклонён без MFA-verified session"
type: TC
tc-type: security
automation: { status: automated, set-version: "1.0" }
verifies:
  - id: SR-08
negative: true
---
## When
POST /v1/privacy/export (session: password-only, no MFA)
## Then
- 403; job не создан; security audit event
```

Аналогичные пары для SR-09 (signed URL valid / expired).

2.6 Контрольные точки качества

Контрольная точка	Доказательства
QG-ADAPT-approve (по смыслу около «архитектурной точки»: `QG-3`)	ADAPT-002 в `approved`, замечания backward закрыты

QG-2 Verification Gate	`TC-080` … `081` успешны; в `last-run` совпадает `set-version` (`1.0`)

2.7 Что дальше

- Полный сценарий под `git` — 03-tool-guide-git
- Compliance mapping — 06-compliance
- Conformance manifest — reference/08
- Интерфейсная часть этого же примера, доведённая до заполненных тел — §7 E8

---

3. E4 — Webhook idempotency (SPEC-API)

Контекст: платёжный провайдер шлёт `POST /webhooks/payment` с `Idempotency-Key`. Дубликаты не должны создавать двойное списание.

T3 (фрагмент): «Повторный webhook с тем же key в течение 24h возвращает тот же `payment_id`, HTTP 200».

SR + SPEC:

```yaml
id: SR-20
type: SR
constrained-by: ["SPEC-API-07"]
```

SPEC-API-07 — контракт: headers, body schema, response codes 200/400/409.

TC (contract pair):

```yaml
id: TC-200
type: TC
tc-type: contract
verifies: [{ id: SR-20 }]
negative: false
```

```yaml
id: TC-201
type: TC
tc-type: contract
verifies: [{ id: SR-20 }]
negative: true
```

Neg: второй key с другим body → 409 Conflict.

Контрольная точка `QG-2`: оба `TC` успешны; в `last-run` совпадает `set-version` (`1.0`).

---

4. E5 — RAG assistant (SPEC-AI)

Контекст: in-app чат отвечает по knowledge base; eval против golden Q&A set.

SPEC-AI-02 — model card, fallback, cost cap.

TC eval (judge ≠ production):

```yaml
id: TC-300
type: TC
tc-type: eval
verifies: [{ id: SPEC-AI-02 }]
judge:
  vendor: "<vendor>"
  model: "eval-judge-v2"          # ≠ production-модели из SPEC-AI-02 (§9.6.2)
baseline:
  artifact: "<golden Q&A dataset>"
  metric-thresholds: {}
negative: false
```

Neg eval: prompt injection в user message → refusal + audit event (`tc-type: eval` , adversarial negative).

См. standard/09 §9.6.2, guide/07 §3.5.

---

5. E6 — Delta-T3: scope dispute

Контекст: после signed TZ клиент просит «добавить экспорт в PDF» — вне scope ADAPT-002.

Правильный путь к источнику истины (SoT):

1. **Не** править неизменяемое T3 и **не** молча расширять SR.
2. Оформить **delta-T3**; состязательный рецензент проводит обзор и выносит вердикт (§7.6.1). При вердикте «findings present» создаётся delta-ADAPT `ADAPT-002-delta-1` (forward + backward); при «no findings» delta-ADAPT не создаётся, а производные артефакты ссылаются на `source.tz-section` delta-T3.
3. Backward finding: «PDF export не входил в Forward §3 ADAPT-002».
4. Решение клиента по находке оформляется **ACTZ** — протоколом уточнения T3 с двусторонней подписью (standard/07 §7.13); находка получает `decided-in: ACTZ-003 §1` .
5. Delta-ADAPT утверждается подписью архитектора (§7.5) → новые SR/SPEC/TC. Итоговое T3 пересчитывается, AT перегенерируются от новой редакции.

Anti-pattern: direct commit в `sr/SR-08` без `delta-ref` — drift class 4.11.6 (standard/04 §4.11).

См. standard/07 §7.6, guide/02-transition-guide.

---

6. E7 — Подсистема как самостоятельный продукт (`implements` -edge)

Контекст: платформа `acme` (система) состоит из подсистемы `acme.notify` — отдельный продукт с собственным бизнес-владельцем (Notify Lead), отдельной командой, отдельным releases-циклом. Сценарий §6.8.2 RENAR Standard.

6.1 Иерархия артефактов

```
acme (система)
├── BR-01 (приём заказов с AI-консультацией)            level: system
├── BR-05 (мониторинг и алерты для операционной службы)  level: system
└── acme.notify (подсистема, самостоятельный продукт)
    └── BR-01 (доставка уведомлений по multichannel)    level: subsystem
         implements:
           - id: BR-01      (раскрывает «приём заказов»: уведомления при
изменениях статуса)
             scope: { system: acme }
           - id: BR-05      (раскрывает «мониторинг»: уведомления о SLA-
нарушениях)
             scope: { system: acme }
         ↓
         SR-01..SR-12 (notify-внутренние требования)
         SPEC-INT-01 (интеграция с acme через message bus)
         SPEC-API-01 (REST API для notify-клиентов)
```

`acme.notify.BR-01` — корневой узел своего дерева требований (`parent` отсутствует, как и у любого BR). Связь с системой выражена **типизированным** ребром `implements[]`, не parent-edge.

6.2 frontmatter `acme.notify/br/BR-01-multichannel-delivery.md` (фрагмент)

```yaml
---
id: BR-01
title: "Доставка уведомлений по multichannel"
type: BR
level: subsystem
scope:
  system: acme
  subsystem: acme.notify

owner: "Notify Lead (notify-side); Architect (acme-side)"

source:
  adapt: ADAPT-NOTIFY-001
  adapt-section: "Forward §2"
  tz-section: "T3-NOTIFY §3"

# === implements-edge §6.8.2 ===
implements:
```

```
  - id: BR-01
    scope:
      system: acme
    rationale: "ADAPT-NOTIFY-001 §2.1 — уведомления при изменении статуса заказа"
  - id: BR-05
    scope:
      system: acme
    rationale: "ADAPT-NOTIFY-001 §2.4 — алерты SLA для operations"

business-context:
  stakeholder: "Notify Lead"
  business-goal: "Своевременная доставка уведомлений end-customer и operations
team"
---

# BR-01: Доставка уведомлений по multichannel

Notify-подсистема доставляет уведомления …
```

6.3 Машиночитаемая trace chain

```
TC-NOTIFY-15
  → verifies SR-NOTIFY-08 (rate-limit для email канала)
        ├─ parent:   acme.notify.BR-01 v1.2
        │              └─ implements: acme.BR-01 v3.0, acme.BR-05 v1.4
        │                                (типизированное межуровневое ребро)
        ├─ source.adapt: ADAPT-NOTIFY-001 §Forward §2.2
        └─ constrained-by: SPEC-API-01, SPEC-OPS-01   (scope: acme.notify)
```

При аудите от TC-NOTIFY-15 восстанавливается полная цепочка: SR → BR подсистемы → BR системы. До v1.0 (без `implements`-edge) цепочка обрывалась на `acme.notify.BR-01` и продолжалась только через текст раздела «Контекст».

6.4 Гейты на стороне носителя

При попытке approve `acme.notify.BR-01` нативный для носителя hook проверяет (см. standard/10 §10.11.1):

1. `acme.BR-01` и `acme.BR-05` существуют в носителе по `id + scope.system`.
2. Оба target BR входят в утверждённую версию комплекта своей системы.
3. Цепочка `acme.notify.BR-01 → acme.BR-01 → …` не образует цикла.
4. `acme.notify.BR-01.level = subsystem` (правило: `implements[]` не применяется на `level: system`).

Если любая проверка проваливается — approve блокируется. Поведение при `acme.BR-01 = deprecated`: warning, не fatal (Architect решает: обновить implements на актуальный BR или отметить `acme.notify.BR-01` как требующий пересмотра).

6.5 Эволюция «модуль → подсистема» через `implements`

Когда модуль приобретает бизнес-владельца (standard/06 §6.9.1):

1. Бизнес-владелец фиксируется через backward finding ADAPT (категория `scope`).
2. После approved delta-ADAPT — модуль повышается до подсистемы; создаётся BR подсистемы.
3. **Новый шаг (v1.0+):** BR подсистемы декларирует `implements[]` на applicable BR системы. Без этого шага сценарий §6.8.2 несёт несовместимое с v1.1 conformance отсутствие traceability.

Anti-pattern: создать `acme.notify.BR-01` с `level: subsystem`, но без `implements[]` — формально допустимо на v1.0 (recommended), но non-conformant на v1.1+. Если родительская система имеет approved BR, отсутствие `implements[]` обязано быть **явно обосновано** в разделе «Контекст» BR со ссылкой на ADAPT§.

См. standard/06 §6.5.2, §6.8.2, §6.10.3, standard/13 §13.3.8.

7. E8 — Интерфейсная часть E3: пример, доведённый до конца

> ***Информативный раздел.*** *Пример не является нормативным текстом и не создаёт обязательств. При расхождении с главами стандарта верны главы. Нормативные требования, которым пример следует, названы в каждом подразделе ссылкой.*

Зачем он здесь. Остальные примеры библиотеки останавливаются на frontmatter и выдержках: они показывают **оболочку** артефакта — идентификаторы, статусы, рёбра графа. Но агент, выводящий реализацию, читает не оболочку, а **утверждение**. E8 продолжает E3 (§2) интерфейсной частью и доводит до конца три тела, которых в корпусе не было ни одного: тело SR (standard/06 §6.6.3), тело SPEC-UI (standard/08 §8.5.6) и пару `ux`-TC (standard/09 §9.6.1).

Что именно он доказывает — и чего не доказывает. Границу лучше провести сразу, до чтения. Пример проверяет **форму** обязательного тела: что разделы §8.4.1 и §8.5.6 заполняются осмысленно, что требование «покрыть каждое доступное через интерфейс действие» операционализуемо, что `ux`-TC пишется по §9.6.1 без изобретения полей. Пример **не** проверяет **объём**: в нём три экрана, и на трёх экранах вывод «требование всех экранов подъёмно на реальной системе» заработать нельзя.

7.1 Граница «всех экранов» проходит по Scope спецификации

Одна частность §8.5.6 обнаруживается только при попытке написать тело, и её стоит назвать до примера.

Формулировка обязательного тела SPEC-UI требует описать «все экраны системы». Буквально прочитанная, она означала бы, что один SPEC-UI обязан описать интерфейс всей системы целиком — от входа до администрирования. Так она не работает и так не задумана: границу описываемого проводит **раздел Scope** того же обязательного тела (standard/08 §8.4.1, пункт 2), а «все экраны» означает **полноту внутри объявленного Scope, без отбора по значимости**.

Обе половины существенны:

- **Scope сужать можно** — он часть обязательного тела и объявляется явно, поэтому не попавшее в него не теряется молча, а видно как граница.

- **Внутри Scope отбирать нельзя** — именно это запрещает положение «Полнота охвата» §8.4.1: не «опишите важные экраны выгрузки», а «опишите все экраны выгрузки».

Ниже Scope объявлен интерфейсом выгрузки персональных данных; экранов в нём три, и описаны все три.

Полнота по системе — перечень экранов. Scope и §8.4.1 дают полноту внутри документа; что сумма всех SPEC-UI покрывает интерфейс целиком, обеспечивают перечень экранов в обязательном теле `SPEC-ARCH-01` системы E3 и правило покрытия (standard/08 §8.5.6.1). В §2.4 `SPEC-ARCH-01` не приводился; здесь — та часть его перечня, которая касается раздела «Приватность»:

Код	Экран	Источник	Покрывающий SPEC-UI
Э-01	Запрос выгрузки	`SR-10`	`SPEC-UI-02`
Э-02	Выгрузка готовится	`SR-10`	`SPEC-UI-02`
Э-03	Запрос отклонён	`SR-10`	`SPEC-UI-02`
Э-04	Подтверждение личности вторым фактором	`SR-08`	`SPEC-UI-01`
Э-05	Выдача переопределения (роль `privacy-officer`)	`SPEC-ARCH-01`, контейнер «Кабинет privacy-officer»	—

Последняя колонка — не часть перечня, а итог проверки при утверждении версии комплекта: перечень ведёт SPEC-ARCH, покрытие объявляет каждый SPEC-UI в `screens[]`. Э-05 в перечне есть, а SPEC-UI на него нет — версия комплекта с таким перечнем **не утверждается** (standard/10 §10.7.2), пока экран не описан либо не исключён из перечня решением, видимым в записи версии. Источник перечня — только описание: `SR-08` и `SR-10` называют экраны, `SPEC-ARCH-01` — контейнер кабинета; маршрут, найденный в реализации и отсутствующий в перечне, — дрейф источника истины и обратная находка, а не основание пополнить перечень задним числом.

7.2 SR-10 — заполненное тело (§6.6.3)

Продолжает `SR-08` из §2.3: тот описывает создание задачи выгрузки со стороны службы, `SR-10` — со стороны интерфейса.

```
# sr/SR-10-export-request-ui.md
id: SR-10
type: SR
parent: { id: BR-02 }
title: "Субъект данных инициирует выгрузку через интерфейс приватности"
source: { adapt: ADAPT-002, adapt-section: "Forward §3", tz-section: "§3.1" }
constrained-by: ["SPEC-UI-02", "SPEC-SEC-03"]
```

Требование

Когда субъект данных подтверждает запрос на экране «Приватность → Выгрузка моих данных», система должна создать задачу выгрузки и перевести пользователя на экран состояния этой задачи.

> *Форма — событийная (standard/06 §6.6.3.1): маркер условия «когда» стоит первым словом и пишется строчной буквой; модальность «должна» — строчная каноническая. Утверждение одно: создание задачи и переход на экран состояния — единый наблюдаемый исход одного действия, а не два независимых требования.*

Поведение

Экран запроса доступен субъекту данных в разделе «Приватность» и показывает состав выгрузки до её заказа: профиль, журнал действий за 730 календарных дней и согласия на маркетинговые рассылки (ТЗ §4.4 в §2.1, уточнение `B-01` из ADAPT-002).

Пока личность субъекта не подтверждена вторым фактором, система должна держать действие «Заказать выгрузку» недоступным и показывать причину недоступности рядом с ним. Подтверждение выполняется существующим механизмом второго фактора; отдельного механизма для выгрузки не вводится.

Если с момента прошлой выгрузки прошло менее 30 дней и переопределение не выдано, то система должна отказать в заказе. Дата, начиная с которой заказ снова возможен, называется субъекту на экране отказа (ТЗ §4.5 в §2.1).

После создания задачи система показывает состояние подготовки и остаётся на экране состояния до появления ссылки на скачивание либо до истечения срока подготовки в 72 часа (ТЗ §4.3 в §2.1). Доставка готового архива нормируется отдельно — `SR-09`.

Каждый заказ, каждый отказ и каждое переопределение записываются в журнал аудита (ТЗ §4.5 в §2.1, уточнение `B-02`).

Ограничения

Срок подготовки — не более 72 часов с момента подтверждения личности. Частота — не чаще одного заказа в 30 дней без переопределения. Переопределение выдаёт участник в роли `privacy-officer`. Полные ограничения — в `SPEC-SEC-03`.

Связь с SPEC

`SPEC-UI-02` нормирует интерфейсную часть: состав экранов, покрытие действий, состояния ожидания и отказа, доступность и языковые версии. `SPEC-SEC-03` нормирует правило частоты, роль выдающего переопределение и срок жизни ссылки на скачивание.

7.3 Пять форм утверждения на одном материале

Standard/06 §6.6.3.1 вводит пять форм. На материале E3 они выглядят так — таблица показывает форму, а не полный текст требований:

Случай	Утверждение
Постоянное	Система должна записывать в журнал аудита каждый заказ выгрузки.
По состоянию	Пока личность субъекта не подтверждена вторым фактором, система должна держать действие «Заказать выгрузку» недоступным.

Событийное	Когда субъект данных подтверждает запрос, система должна создать задачу выгрузки.
Нежелательное поведение	Если с момента прошлой выгрузки прошло менее 30 дней, то система должна отказать в заказе.
Опциональная возможность	При наличии выданного переопределения система должна принять заказ раньше 30 дней.

Три частности формы видны на этих же строках: маркер «при наличии» вместо кальки «где»; обязательное «то» после «если»; одно утверждение в одном предложении — там, где хотелось написать «отказать и назвать дату», отказ и сообщение разделены на два предложения, потому что склеенное утверждение верифицируется неделимо и один TC доказывал бы половину.

7.4 SPEC-UI-02 — заполненное обязательное тело (§8.4.1 + §8.5.6)

```yaml
# specs/ui/SPEC-UI-02-privacy-export.md
id: SPEC-UI-02
type: SPEC
spec-type: SPEC-UI
title: "Интерфейс выгрузки персональных данных"
source: { adapt: ADAPT-002, adapt-section: "Forward §3" }
depends-on: ["SPEC-SEC-03", "SPEC-API-04"]
screens: ["Э-01", "Э-02", "Э-03"]     # коды из перечня экранов SPEC-ARCH-01 (§7.1)
ui-platform: web
target-users: [{ role: data-subject, persona: "ADAPT-002 §2.1" }]
design-system: "AcmeCRM Design Kit 4"
accessibility-level: WCAG-AA
i18n: required
mockup-links: [{ tool: figma, url: "<ссылка на макет>", version: "v3" }]
baseline-images:
  - "ai-concepts/baselines/SPEC-UI-02-screen-01.png"
  - "ai-concepts/baselines/SPEC-UI-02-screen-02.png"
  - "ai-concepts/baselines/SPEC-UI-02-screen-03.png"
```

Назначение

Спецификация нормирует интерфейс, которым субъект данных заказывает выгрузку своих персональных данных, наблюдает за её подготовкой и получает отказ с объяснимой причиной. Она даёт агенту, реализующему интерфейс, единственный источник истины о составе экранов и о том, что на них доступно; всё, что здесь не описано, источника истины не имеет и домысливанию не подлежит (standard/02 §2.3.3).

Scope

Входит: три экрана раздела «Приватность → Выгрузка моих данных» — запрос выгрузки, состояние подготовки, отказ; все действия, доступные субъекту данных на этих экранах; сквозные элементы этих экранов.

Не входит: экран подтверждения второго фактора (Э-04 перечня: правило — `SPEC-SEC-03`, экран — `SPEC-UI-01`; переиспользуется без изменений); интерфейс участника в роли `privacy-officer`, выдающего переопределение (отдельная спецификация, в E8 не рассматривается); почтовое уведомление о готовности архива (не экран).

Общая структура интерфейса

Раздел «Приватность» — вкладка в настройках учётной записи. Выгрузка — один из её пунктов; экраны выгрузки образуют линейную последовательность без ветвлений: запрос → состояние, либо запрос → отказ. Возврат в настройки доступен с любого из трёх экранов. Всплывающих окон в разделе нет: отказ и состояние — полноценные экраны, потому что оба адресуемы ссылкой и оба обязаны переживать перезагрузку страницы.

Экраны

Экранов три; описаны все три.

Код	Экран	Когда показывается
Э-01	Запрос выгрузки	Субъект открыл пункт «Выгрузка моих данных»; активной задачи нет
Э-02	Выгрузка готовится	Задача создана и не завершена, либо завершена и ссылка ещё жива
Э-03	Запрос отклонён	Заказ отклонён правилом частоты или неподтверждённой личностью

Э-01 «Запрос выгрузки». Содержит: заголовок раздела; перечень того, что войдёт в архив, — профиль, журнал действий за 730 дней, согласия на рассылки; строку о сроке подготовки до 72 часов; строку о частоте не чаще раза в 30 дней; отметку о подтверждении личности с текущим состоянием; действие «Заказать выгрузку»; действие «Вернуться в настройки».

Э-02 «Выгрузка готовится». Содержит: состояние подготовки словами, без доли выполнения в процентах — доля на стороне интерфейса неизвестна; время создания задачи; предельный срок готовности; действие «Обновить состояние»; действие «Отменить выгрузку»; действие «Вернуться в настройки». После появления ссылки на том же экране появляется действие «Скачать архив» и срок жизни ссылки.

Э-03 «Запрос отклонён». Содержит: причину отказа обычным языком; дату, начиная с которой заказ снова возможен, — для отказа по частоте; действие «Запросить переопределение»; действие «Вернуться в настройки». Для отказа по неподтверждённой личности вместо даты показывается действие «Подтвердить личность».

Пути пользователя и покрытие действий

Покрытие проверяется поимённо: каждое действие каждого экрана названо ниже хотя бы раз (standard/08 §8.5.6 — «покрывающие каждое доступное через интерфейс пользовательское действие»).

Путь	Экраны	Покрытые действия
П-1. Обычный заказ	Э-01 → Э-02	Раскрытие перечня состава архива, «Заказать выгрузку», «Обновить состояние», «Скачать архив»
П-2. Заказ до подтверждения личности	Э-01 → Э-03 → Э-01	«Подтвердить личность»

П-3. Повторный заказ раньше срока	Э-01 → Э-03	«Запросить переопределение»
П-4. Отказ от начатой выгрузки	Э-02 → Э-01	«Отменить выгрузку»
П-5. Выход из раздела	Э-01, Э-02, Э-03	«Вернуться в настройки» с каждого из трёх экранов

Сверка счётом: действий на трёх экранах восемь — раскрытие перечня состава архива, «Заказать выгрузку», «Вернуться в настройки», «Обновить состояние», «Отменить выгрузку», «Скачать архив», «Запросить переопределение», «Подтвердить личность». Все восемь названы в путях П-1…П-5. Непокрытых действий нет.

Сквозные элементы

Права доступа. Все три экрана доступны только владельцу учётной записи. Участник в роли `privacy-officer` этих экранов не видит и заказать выгрузку за субъекта не может; его интерфейс — вне Scope.

Уведомления. Интерфейс не уведомляет о готовности сам: уведомление уходит почтой. На Э-02 это сказано явно, чтобы субъект не держал вкладку открытой.

Состояния ошибки. Отказ по правилам — это Э-03, полноценный экран. Сбой связи и недоступность службы выгрузки показываются на том экране, где произошли, сообщением с действием «Повторить»; они не переводят на Э-03, потому что заказ не отклонён, а не доставлен.

Пустые состояния. У Э-01 пустого состояния нет — перечень состава непуст всегда. На Э-02 пустое состояние невозможно по построению: экран показывается только при существующей задаче.

Тон и стиль

Обращение на «вы» со строчной буквы. Причина отказа называется прямо и без извинений: «Выгрузка доступна раз в 30 дней. Следующий заказ — с 12 июня». Служебные слова — «задача», «архив», «журнал действий» — не заменяются техническими: `job`, `bundle`, `activity log` на экранах не появляются.

Доступность

Уровень — WCAG-AA. Существенно для этих трёх экранов: смена состояния на Э-02 объявляется программе экранного доступа как живая область, иначе субъект не узнает о готовности архива; причина недоступности действия «Заказать выгрузку» связана с самим действием программно, а не только расположена рядом; отказ на Э-03 не передаётся одним лишь цветом; порядок обхода с клавиатуры совпадает с порядком чтения на всех трёх экранах.

Языковые версии

Обязательны русская и английская. Дата возможного повторного заказа выводится в языковом формате, а не одним видом для обеих версий. Длина строки причины отказа не ограничена шириной кнопки: английская формулировка длиннее русской, и экран обязан переносить её, а не обрезать.

Связь с требованиями

`SR-10` — заказ выгрузки через интерфейс (`constrained-by`). `SR-09` — доставка готового архива, интерфейсная часть которой ограничена действием «Скачать архив» на Э-02. `BR-02` —

деловое требование, из которого оба выведены.

Связь с другими SPEC

`SPEC-SEC-03` — правило частоты, роль выдающего переопределение, срок жизни ссылки; интерфейс эти правила показывает, но не определяет. `SPEC-API-04` — источник состояний задачи, отображаемых на Э-02.

Verification

`TC-090` и `TC-091` (§7.5). Доступность проверяется отдельными TC типа `accessibility`, языковые версии — TC типа `i18n`; в E8 они не развёрнуты.

7.5 Пара `ux` -TC (§9.6.1)

Двухслойная структура §9.6.1: слой намерения исполняет управляющий агент, слой перцептивной проверки — рецензирующая модель. Обязательное расширение frontmatter — `judge.vendor`, `judge.model`, `baseline.artifact`, `baseline.perceptual-diff-threshold`. Секции тела — по standard/09 §9.4, включая обязательный «Out of scope».

```yaml
---
id: TC-090
title: "Субъект заказывает выгрузку и видит экран подготовки"
type: TC
tc-type: ux
automation: { status: automated, set-version: "1.0" }
verifies:
  - id: SR-10
    statement: 1                  # SR-10#1 — «Требование», событийная форма
negative: false
judge: { vendor: google, model: "gemini-2.5-pro" }
baseline:
  artifact: "ai-concepts/baselines/SPEC-UI-02-screen-02.png"
  perceptual-diff-threshold: 0.03
---
```

Контекст. `SR-10`, раздел «Требование»: «Когда субъект данных подтверждает запрос … система должна создать задачу выгрузки и перевести пользователя на экран состояния этой задачи». Экран Э-02 по `SPEC-UI-02`.

Предусловия. Учётная запись субъекта с подтверждённым вторым фактором; активных задач выгрузки нет; прошлая выгрузка была более 30 дней назад. Состояние подготовлено механизмом заполнения данных.

Шаги. Намерение, не селекторы: субъект данных хочет получить архив своих данных после входа в раздел приватности.

Pass-критерий. На отрисованном состоянии виден экран подготовки: состояние названо словами, показаны время создания задачи и предельный срок готовности, доступно действие обновления состояния. Перцептивное расхождение с эталоном не превышает 0.03.

Fail-критерий. Наблюдаемые признаки нарушения: остались на экране запроса; доля выполнения показана в процентах; предельный срок отсутствует либо показан как «неизвестно»; ссылка на скачивание предъявлена до готовности архива; задача создана дважды при одном подтверждении.

Постусловия. Ровно одна задача выгрузки в состоянии подготовки; запись в журнале аудита создана. Задача снимается механизмом очистки после прогона.

Out of scope. Не проверяется доставка готового архива и срок жизни ссылки — покрыто парой TC для `SR-09`. Не проверяется отказ по правилу частоты — покрыто `TC-091`. Не проверяется доступность экрана — покрыто TC типа `accessibility`.

```yaml
---
id: TC-091
title: "Повторный заказ раньше 30 дней отклонён с названной датой"
type: TC
tc-type: ux
automation: { status: automated, set-version: "1.0" }
verifies:
  - id: SR-10
    statement: 5                  # SR-10#5 — «Поведение», отказ по сроку
negative: true
judge: { vendor: google, model: "gemini-2.5-pro" }
baseline:
  artifact: "ai-concepts/baselines/SPEC-UI-02-screen-03.png"
  perceptual-diff-threshold: 0.03
---
```

Контекст. `SR-10`, раздел «Поведение»: «Если с момента прошлой выгрузки прошло менее 30 дней и переопределение не выдано, то система должна отказать в заказе». Экран Э-03 по `SPEC-UI-02`.

Предусловия. Учётная запись субъекта с подтверждённым вторым фактором; прошлая выгрузка завершена 5 дней назад; переопределение не выдавалось.

Шаги. Субъект данных хочет заказать выгрузку повторно после недавно полученного архива.

Pass-критерий. На отрисованном состоянии виден экран отказа: причина названа обычным языком, указана дата возможного следующего заказа, доступно действие запроса переопределения. Перцептивное расхождение с эталоном не превышает 0.03.

Fail-критерий. Наблюдаемые признаки нарушения: задача выгрузки создана вопреки правилу; отказ показан без даты следующего заказа либо с датой в прошлом; отказ передан только цветом, без текста; отказ показан всплывающим сообщением вместо экрана; действие запроса переопределения отсутствует; причина изложена служебным текстом сбоя.

Постусловия. Новых задач выгрузки нет; в журнале аудита создана запись об отказе.

Out of scope. Не проверяется выдача переопределения участником в роли `privacy-officer` — его интерфейс вне Scope `SPEC-UI-02`. Не проверяется отказ по неподтверждённой личности — отдельный TC той же пары форм.

Эталоны. Оба TC сверяются с эталонными изображениями из `baseline-images` спецификации. Замена эталона проходит механизм утверждения с отметкой `[baseline-`

`update]` (standard/09 §9.13); автоматическое обновление запрещено, иначе изменившийся интерфейс сам себя объявит верным.

7.6 SPEC-UC-01 — сценарий использования (§8.5.12)

Сценарий сшивает `SR-10`, `SPEC-UI-02` и `SPEC-API-01`; исполнитель — человек, поэтому каждый шаг покрыт `ux`-TC, а не `system`-TC в обход интерфейса (standard/09 §9.8).

```
---
id: SPEC-UC-01
title: "Заказ выгрузки персональных данных субъектом"
type: SPEC-UC
spec-type: SPEC-UC
level: system
role: human
persona: "ADAPT-003 §Persona: субъект данных"
covers: [SR-10, SPEC-UI-02, SPEC-API-01]
entry: SPEC-UI-02
steps:
  - { n: 1, action: "Открывает экран запроса выгрузки", ref: SPEC-UI-02, expects:
"Экран Э-01 с формой запроса" }
  - { n: 2, action: "Подтверждает запрос", ref: SR-10#1, expects: "Создана задача
выгрузки; переход на Э-02" }
  - { n: 3, action: "Ожидает подготовки", ref: SPEC-UI-02, expects: "Э-02
показывает состояние задачи" }
  - { n: 4, action: "Получает отказ при повторном заказе раньше 30 дней", ref:
SR-10#5, expects: "Э-03 с причиной отказа" }
source: { adapt: "ADAPT-003", adapt-section: "Forward §4", tz-section: "§5.2" }
---
## Цель и исполнитель
Субъект данных заказывает выгрузку своих данных и видит состояние задачи.

## Предусловия
Личность подтверждена; предыдущей выгрузки не было либо прошло ≥ 30 дней.

## Основной путь
Шаги 1–3 из frontmatter; каждый ссылается на утверждение `SR-10` или экран `SPEC-
UI-02`.

## Альтернативные и негативные ветви
Шаг 4 — отказ по сроку (`SR-10`, «Поведение»); отказ по неподтверждённой личности
— вне Scope (см. `SPEC-UI-02`).

## Постусловия
Задача выгрузки существует в состоянии «в подготовке» либо отказ зафиксирован в
журнале.

## Покрывающие TC
`TC-090` (шаги 1–3, positive), `TC-091` (шаг 4, negative); оба `tc-type: ux`.
```

Шаг без `ref` — нарушение структурной полноты (§8.5.12.1): версия комплекта с таким сценарием не утверждается.

7.7 Что этот пример не доказывает

Отмечено явно, чтобы пример не читался как больше, чем он есть.

- **Подъёмность «всех экранов» на реальной системе.** Экранов три. На трёх экранах проверена форма обязательного тела, а не его объём. Продукт с полусотней экранов даст другую цену, и она этим примером не измерена.
- **Пригодность формы утверждения на накопленном описании.** Пять форм §6.6.3.1 применены к пяти утверждениям, написанным сразу по форме. Перевод существующего описания в эти формы — другая задача, и модальность правила остаётся рекомендательной именно поэтому (standard/06 §6.6.3.1).
- **Машинная проверка покрытия перечня экранов.** Таблица §7.1 сведена вручную; статический TC, сверяющий `screens[]` всех SPEC-UI с перечнем `SPEC-ARCH-01`, в примере не написан.
- **Работоспособность перцептивной проверки.** `TC-090` и `TC-091` написаны по §9.6.1, но не прогонялись: эталонных изображений в корпусе нет, и рецензирующая модель к примеру не подключалась. Проверено, что поля §9.6.1 заполняются без изобретения новых, — не более.

Guide RENAR 1.1 — renar.tech

10. Миграция на v1.0

> ***Историческое руководство v0 → v1.0.*** *Поля* `requirement-version`*, статусы артефактов и переходы* `approved → verified` *ниже описывают v1.0; в v1.1 они заменены версией комплекта описания (* `set-version`*, standard/10 §10.5) — руководство по миграции v1.0 → v1.1 выходит вместе с v1.1.*
>
> *Для команд, которые уже вели требования «по-своему» или на ранних черновиках RENAR. Цель — **без big-bang**: сохранить неизменяемые ID, заменить deprecated конструкции, выпустить новый conformance manifest. Нормативная база — standard/04 §4.14, CHANGELOG §Migration.*

Не путать с 02-transition-guide — там поэтапный вход в RENAR-1..5 с нуля; здесь — **breaking rename** и выравнивание схемы при переходе на текущую редакцию стандарта.

1. Когда нужна эта миграция

Ситуация	Действие
Проект без RENAR, но есть Т3 + Jira	Сначала 02-transition-guide, не этот документ
Файлы с `INT-SR`, `INT-TC`, `AIC`, `UIC`, `TS`	Это руководство
`verifies[].version` без `requirement-version`	Обновить TC frontmatter (reference/02 §8)
Manifest `renar-version: 0.1-draft` или отсутствует	Выпустить новый manifest (reference/08)

2. Таблица замен типов (closed list v1.0)

Deprecated	Canonical	Шаг миграции
`INT-SR`	`SR` + `constrained-by: [SPEC-INT-N]`	Переименовать type; создать/привязать SPEC-INT
`INT-TC`	`TC` + `tc-type: contract`	Добавить `type: TC`, `tc-type: contract`
`AIC`	`SPEC-AI`	Перенести body в SPEC-AI frontmatter
`UIC`	`SPEC-UI`	Перенести baselines в `specs/ui/baselines/`
`TS`	`SPEC-ARCH` или `SPEC-OPS`	По содержанию (архитектура vs ops runbook)
`TM` (module SR type)	`SR` + `level: module`	Убрать pseudo-type, задать level

`tc-type: acceptance`	`tc-type: business`	Переименовать значение. Приёмок теперь две: `business` -TC меряет бизнес-цель BR (внутренний контур), а соответствие контракту проверяет `AT` (standard/09 §9.19) — одно имя на два разных предмета убрано (§9.5)
`approval.client-signature` на ADAPT	Подпись только архитектора	Убрать клиентскую подпись из ADAPT; решения клиента перенести в подписанные `ACTZ` и проставить `decided-in` на находках (standard/07 §7.5, §7.13)
Состояние ADAPT `client-ready`	`asked`	Изъято из машины состояний: вынесение вопросов клиенту — предмет ACTZ, а не состояние ADAPT (standard/10 §10.8)

ID не менять. Если filename содержит legacy prefix — допустимо поле `legacy-id` во frontmatter (informative traceability).

---

3. Пошаговый план (1–2 спринта)

Фаза A — инвентаризация (1–2 дня)

1. `grep` / search по носителю: `INT-SR`, `INT-TC`, `AIC`, `UIC`, `type: system` (ошибочный TC type).
2. Список артефактов с broken `verifies` / missing `source.adapt`.
3. Зафиксировать текущий `RENAR-CONFORMANCE.yaml` (если есть) как опорное состояние.

Фаза B — проход по схеме (3–5 дней)

1. Batch-rename типов по таблице §2 (один PR на тип или один atomic change-set на delta-T3).
2. TC: `type: TC`, `tc-type`, `verifies[].requirement-version`, `last-run.requirement-version`.
3. SR: `constrained-by[]` на все используемые SPEC.
4. BR: источник адаптации — `source.adapt` на approved ADAPT либо `source.adversarial-review-ref` на issued AR при вердикте «no findings»; `source.tz-section` обязателен всегда (standard/07 §7.4.1).

Фаза C — валидация (1–2 дня)

1. Hooks носителя / CI: frontmatter validator (reference/02).
2. Прогнать TC; обновить `last-run`.
3. Чек-лист самооценки — reference/08 §2.

Фаза D — manifest (1 день)

1. Bump `renar-version: "1.0"`.
2. Increment `manifest-version`; новый `manifest-id`.

3. Подпись Architect / Tech Lead (V6).

4. Частые ловушки

Ошибка	Последствие	Исправление
Переименовать `SR-05` → `SR-05-v2`	Нарушение V1 immutable ID	`deprecated` + новый ID с `replaces`
Оставить `type: system` на TC	Validator fail; KG drift	`type: TC + tc-type: system`
Мигрировать код раньше `.req`	SoT inversion нарушена	Сначала SR/SPEC/TC approved, потом TR
Skip состязательного обзора «только для новых ТЗ»	Non-conformant для legacy TZ	Ретроспективный состязательный обзор каждого активного ТЗ; ADAPT — только при вердикте «findings present» (standard/07 §7.4.1)

5. Откат

Миграция идёт через историю носителя (V1). Откат — revert change-set / PR, **не** delete артефактов. Deprecated артефакты остаются с `status: deprecated` для audit.

6. Связанные документы

Документ	Зачем
02-transition-guide	RENAR-1..5 без schema breaking changes
09-worked-examples	Эталонные frontmatter после миграции
reference/07	Внешнее заявление ISO 29148
standard/13 §13.7	Re-assessment после миграции

Guide RENAR 1.1 — renar.tech

11. Работа с ИИ-агентом

> ***Это не «ручной режим».*** *Распространённое заблуждение: раз RENAR требует утверждений человеком, значит всё делается вручную. На деле наоборот — основную работу (черновики `BR` / `SR` / `SPEC` / `TC`, первичную интерпретацию ТЗ, поиск противоречий) выполняет агент. Человек не печатает артефакты, а* ***утверждает*** *их в нескольких фиксированных точках. Эта глава показывает, как загрузить стандарт в агента, какие команды давать и что вы получите на выходе.*
>
> *Нормативная модель делегирования — standard/05 Роли и точки утверждения ADAPT — standard/07 §7. Самодостаточная редакция для агента — `RENAR-AGENT-RU.md` (в корне репозитория и на renar.tech).*

1. Кто что делает: карта делегирования

Агент — исполнитель: он производит и ведёт артефакты, но не владеет ими и не ставит подписей. Человек — владелец и тот, кто утверждает. Граница проходит по фиксированным точкам:

Шаг	Делает агент	Утверждает человек
Импорт ТЗ	Регистрирует ТЗ как неизменяемый документ, фиксирует подпись клиента	— (подпись ставит клиент при сдаче ТЗ)
Состязательный обзор	Агент-критик на **другой модели** выносит вердикт «есть находки» / «находок нет»	Архитектор фиксирует вердикт как свидетельство
ADAPT	Готовит черновик: прямая интерпретация + обратные находки	**Подпись Архитектора** — и только его: ADAPT клиенту не выносится
ACTZ	Готовит проект протокола уточнения ТЗ по находкам	**Двусторонняя подпись:** клиент + исполнитель. Вопросы перед выносом агрегирует и переформулирует Архитектор
Декомпозиция	Создаёт `BR` → `SR` → `SPEC` → `TR` с провенансом	Архитектор одобряет переход в `approved` (QG-0)
Тесты	Пишет пары `TC` (позитивный + негативный) — **до** реализации и **не** тем агентом, который её пишет	Автоматический прогон фиксирует результат (QG-2)
Приёмочные тесты	**Изолированный** агент на другой модели выводит `AT` только из итогового ТЗ	Релизный гейт: все `AT` зелёные
Сдача	Предъявляет бизнес-результат	Заинтересованная сторона принимает его (QG-4)

Если утверждение человеком не получено, артефакт остаётся в `draft`, а переходы блокирует носитель. Агент никогда не объявляет себя владельцем и не подписывает.

Граница двух контуров. Всё, что показано клиенту и подписано, — обязательство (ACTZ). Всё, что не показано, — интерпретация (ADAPT). Агент обязан различать: вопрос, требующий решения клиента, идёт в проект ACTZ, а не «додумывается» в прямой интерпретации (standard/07 §7.13).

1.1 Роли производственной модели исполнителя ↔ роли RENAR

Организация-исполнитель обычно ведёт собственный список ролей. RENAR его не переопределяет (standard/05 §5.2.2): достаточно сопоставить каждую роль с закрытым списком §5 и с зоной подписи. Пример сопоставления для производственной модели с семью ролями:

Роль исполнителя	Зона ответственности	Роль RENAR (§5.2.1)	Подписывает
Руководитель проектов	портфель заказов и мощности	Супервизор	—
Главный архитектор	портфель архитектуры, стеки, правила проверки	Архитектор (нормы уровня организации; ужесточения — `declared-stricter`, §5.7.2)	правила стека, типовые решения
Куратор клиента	интерфейс с заказчиком	Уполномоченное лицо исполнителя (§5.4)	`vendor-signature` ACTZ; организует подпись заинтересованной стороны
Системный архитектор заказа	архитектура заказа, комплект описания	Архитектор — owner ADAPT, SPEC, версии комплекта (§5.3)	ADAPT, версия комплекта, протоколы, стенды (SPEC-TEST)
Инженер-супервизор разработки	парк агентов, ход разработки	Супервизор; приёмка задачи — при запрете совмещать исполнение и приёмку одной задачи (§5.5.5)	приёмка задач
ИИ-инженер	профили агентов, промпты	Супервизор (профили AI-агента; `ai-provenance`, standard/04 §4.10.1)	—
Администратор платформы	инфраструктура	не роль RENAR — обеспечивает возможности носителя V1–V6 (standard/03)	—

Заказчик — Заинтересованная сторона (`client-signature`); агент — AI-агент: производит артефакты, не владеет ими и не подписывает; оператор — Супервизор. Одно лицо может совмещать роли, и это называется, а не компенсируется; не совмещаются только исполнение и приёмка одной задачи и две стороны подписи ACTZ.

2. Как загрузить стандарт в агента

Агенту не нужен весь нормативный корпус целиком — для повседневной работы достаточно одного файла.

1. Скачайте `RENAR-AGENT-RU.md` — самодостаточную операционную редакцию (≈400 строк): карта артефактов, предусловия артефактов, ADAPT, закрытые списки, жёсткие правила, формы записи (frontmatter + тела разделов).
2. Положите файл рядом с агентом — в контекст проекта, в системный промпт или как вложение сессии (зависит от вашего инструмента).
3. Дайте установочную команду: «Изучи этот файл — это стандарт RENAR. Дальше веди инженерию требований строго по нему».
4. По желанию добавьте проектный контекст: реестр систем и подсистем, ссылку на ваш носитель артефактов, конвенцию именования.

Полный корпус (`standard/` , `reference/`) подключайте только когда возник формальный спор о точной формулировке нормы — для большинства задач хватает операционной редакции.

---

3. Как давать команды

Команды агенту — это обычные формулировки на естественном языке, привязанные к артефакту, который агент производит. Ниже — типовые примеры; подставьте свои идентификаторы.

Импорт и обзор ТЗ:

```
Вот ТЗ клиента (файл TZ-2026-001). Зарегистрируй его как неизменяемый документ.
Затем проведи состязательный обзор: вынеси вердикт «есть находки» или «находок
нет»,
с обоснованием по 7 категориям (противоречие, пропуск, скрытое предположение,
реализуемость, регуляторика, терминология, граница работ).
```

Создание ADAPT (если есть находки):

```
Обзор нашёл находки. Подготовь черновик ADAPT: прямая интерпретация по разделам
ТЗ
плюс обратные находки. Не додумывай за клиента — то, что неясно, выноси в
находку.
ADAPT — внутренний документ, клиенту он не уходит.
```

Протокол уточнения ТЗ (ACTZ):

```
Собери находки, требующие решения клиента, в проект ACTZ — «Протокол уточнения ТЗ
№ 1».
Каждый пункт формулируй на языке обязательств («домен сравнивается без учёта
регистра»),
а не толкования, и добавь предложение исполнителя. Статус — draft: выносить
```

```
клиенту
и подписывать буду я.
```

Декомпозиция:

```
ACTZ-001 подписан обеими сторонами, ADAPT утверждён. Проставь decided-in на
пункты
протокола во всех закрытых находках. Выведи из ADAPT BR (бизнес-цель), затем SR
(поведение системы), затем SPEC нужных типов. У каждого артефакта проставь source
и parent. Покажи граф связей перед тем, как фиксировать.
```

Тесты:

```
Для каждого нормативного утверждения в SR-03 напиши пару TC: позитивный и
негативный.
Pass-критерий — бинарный и наблюдаемый. Реализации ещё нет и быть не должно:
сначала
заморозим тесты, потом код — и писать код будет другой агент.
```

Приёмочные тесты (отдельная сессия, другая модель):

```
Вот итоговое ТЗ: TZ-2026-001 плюс подписанные ACTZ-001 и ACTZ-002. Больше у тебя
ничего нет и не будет — ни требований, ни спецификаций, ни тестов, ни кода.
Выведи AT: приёмочные тесты по пунктам контракта, с дословной цитатой пункта
рядом с шагами проверки, пары позитивный/негативный.
```

Хорошая команда называет **вход** (какой артефакт), **действие** (что вывести) и **ожидание** (что проверить). Агент сам решает рутину; вмешивайтесь там, где нужно ваше решение.

---

4. Что получается на выходе

Агент возвращает не текст в чате, а готовые артефакты носителя — с машиночитаемым frontmatter и телом по фиксированным разделам:

- **`BR`** — бизнес-потребность: кто, что, зачем; критерии успеха; контекст со ссылкой на ADAPT.
- **`SR`** — поведение системы: одно требование, поведение, ограничения, связь со `SPEC`.
- **`SPEC-<ТИП>`** — спецификация одного из двенадцати типов, связанная с `SR` через `constrained-by[]`.
- **`TC`** — пары позитивный/негативный с `verifies[]` на проверяемый артефакт.
- **`ACTZ`** — проект протокола уточнения ТЗ: пункты решений на языке обязательств, `status: draft` до выноса клиенту.
- **`AT`** — приёмочные тесты со ссылками только на ТЗ и ACTZ, с дословной цитатой пункта контракта и провенансом изолированного генератора.

У каждого артефакта проставлен `source` (происхождение от ТЗ или ADAPT) и, для AI-сгенерированных, блок `ai-provenance` (модель, шаблон промпта, факт человеческой правки). Готовые copy-paste заготовки всех форм — в reference/12 Шаблоны документов.

Признак хорошего выхода: по любому артефакту прослеживается цепочка вверх — до `SR`, до `BR`, до раздела ТЗ. Если цепочка рвётся, артефакт не готов.

---

5. Точки человеческого утверждения

Делегирование агенту широкое, но у него есть жёсткие границы. Человек обязателен в пяти местах (нормативно — standard/10 Жизненный цикл и контрольные точки):

1. **Подпись ТЗ** — клиент подписывает договорной вход; агент только регистрирует.
2. **Двусторонняя подпись ACTZ** — клиент и исполнитель подписывают протокол уточнения ТЗ. Это единственное место, где возникает обязательство перед клиентом (§7.13).
3. **Подпись Архитектора на ADAPT (QG-3)** — Архитектор подтверждает, что все находки отработаны (каждая несёт `decided-in` на пункт подписанного ACTZ) и что интерпретация реализуема. Клиент ADAPT не подписывает: он его не видел (§7.5).
4. **Одобрение QG-0** — Архитектор переводит `BR` / `SR` / `SPEC` из `draft` в `approved`. Только после этого разрешена декомпозиция дальше.
5. **Приёмка** — релизный гейт по AT (все зелёные) и, опционально, QG-4 по бизнес-результату.

Между этими точками агент действует сам. Попытка заставить агента подписать или объявить его ответственным (`Accountable`) — нарушение модели ролей.

---

6. Кто пишет тест и почему он обязан быть красным

Это единственная часть работы, где агенту прямо **запрещено** делать всё сразу — и запрет содержательный, а не церемониальный.

Изоляция авторства (standard/09 §9.18). Тест, написанный тем же агентом, который пишет код, — и особенно написанный **после** кода, — честно проверяет то, что написано. Но проверять он должен не код, а требование. Поэтому изоляция держится на трёх осях:

Ось	Что означает на практике
Время	`TC` заморожен в `ready` до старта задачи. Носитель не даёт открыть TR, пока тесты не заморожены
Автор	Агент, пишущий `TC`, — не тот, что пишет реализацию: другая сессия или другая модель
Изменение	Правку Pass/Fail-критериев замороженного теста утверждает инженер; молча «подправить тест под код» нельзя

Красная история (§9.18.2). Тест, ни разу не наблюдавшийся красным, доказательством не является. Фиксирующий прогон делается **до** реализации, и тест обязан упасть: проверяемой функциональности ещё нет. Зелёный на фиксирующем прогоне — не успех, а сигнал разбора: либо тест ничего не проверяет, либо функциональность уже есть.

Исключение одно — тесты класса `implementation-originated` (standard/06 §6.13): внутренняя техническая деталь, добавленная агентом при реализации (защитная проверка,

журналирование), не наблюдаемая клиентом. Такой тест пишется после кода и рождается зелёным — красной истории у него не бывает по построению. Компенсация обязательна: его зубастость доказывается **убитым мутантом**. И граница класса жёсткая: наблюдаемое клиентом поведение через него не легализуется — обязательство перед клиентом не может возникнуть из кода.

Изоляция генератора AT (§9.19.2). Приёмочные тесты выводит отдельный агент на другой модели, которому на вход подаётся **только** итоговое ТЗ. Доступ к ADAPT, `BR` / `SR` / `SPEC`, `TC` и коду запрещён — не из гигиены, а потому что AT есть симуляция взгляда заказчика. Агент, увидевший ADAPT, воспроизведёт ту же интерпретацию, и AT начнёт подтверждать её вместо контракта: уровень приёмки схлопнется в уровень верификации.

---

7. Типичный сеанс целиком

Чтобы снять ощущение «магии», вот сквозной сценарий — что делает агент и где вступает человек. Порядок шагов — иллюстрация: агентская редакция задаёт предусловия артефактов (её раздел 4), а порядок работ — производственная модель вашей организации (standard/01 §1.2.1):

1. Вы передаёте агенту ТЗ и `RENAR-AGENT-RU.md`. Агент регистрирует ТЗ.
2. Агент-критик (другая модель) проводит состязательный обзор → вердикт «есть находки».
3. Агент готовит черновик ADAPT с находками и проект ACTZ. **Архитектор** доводит протокол до вида, понятного клиенту, выносит его, подписывает вместе с клиентом → ACTZ подписан; находки получают `decided-in`; **Архитектор** утверждает ADAPT своей подписью.
4. Агент выводит `BR` → `SR` → `SPEC`, показывает граф связей. **Архитектор** одобряет QG-0.
5. Агент-тестировщик пишет пары `TC`; фиксирующий прогон — красный; тесты замораживаются.
6. **Другой** агент пишет реализацию, пока тесты не позеленеют; runner фиксирует результат (QG-2).
7. Изолированный агент перед испытаниями выводит `AT` из итогового ТЗ и прогоняет их. Все зелёные → продукт можно предъявлять.
8. Агент собирает бизнес-результат. **Заинтересованная сторона** принимает (QG-4).

Человек включается для утверждения, не для печати артефактов, — но включается обязательно, и подпись под обязательством ставит только он.

---

8. Частые ошибки

Ошибка	Почему плохо	Как правильно
Просить агента «просто написать код» по ТЗ	Теряется источник истины — код без требований	Сначала требования, потом реализация из них
Пропустить состязательный обзор	«ADAPT не нужен» становится молчаливым допущением	Обзор обязателен всегда; «находок нет» — это зафиксированный вердикт другой модели
Позволить агенту подписать ADAPT или ACTZ	Агент не владелец и не подписант	Подпись — всегда человек: ADAPT подписывает Архитектор, ACTZ — клиент и исполнитель

Считать находку закрытой без подписанного ACTZ	Интерпретация опирается на решение, которого клиент не принимал	`resolved` только с `decided-in` на пункт подписанного протокола
Отправить клиенту сырой вывод агента как протокол	Клиент подписывает то, чего не понимает	Вопросы агрегирует и переформулирует Архитектор — на языке обязательств
Реконструировать `SR` из готового кода	Инверсия источника истины	Менять требование → затем код; исключение — обоснованный bug-fix
Дать одной модели и генерировать, и проверять	Нет независимости обзора	Критик — на другой модели, чем генератор
Один агент пишет и код, и тест к нему	Тест проверяет код, а не требование	Разные агенты; тест заморожен до реализации и хоть раз был красным
Дать генератору AT «для контекста» посмотреть SR или ADAPT	AT начнёт подтверждать интерпретацию вместо контракта	На вход — только итоговое ТЗ; другая модель; никакого внутреннего контура
Прогонять на испытаниях AT, выведенные полгода назад	Система предъявляется против контракта, которого уже нет	AT перегенерируются перед каждыми испытаниями

← К обзору руководства

12. Миграция на v1.1

> *Шаг 4 процедуры §13.9.3 для волны v1.1 (ADR-015…ADR-026, GitLab #30–#54). Для внедрений, заявивших соответствие RENAR v1.0. Выпуск minor-версии — немедленный триггер повторной оценки (§13.7.3): заявление v1.0 действует до переоценки, но не переносится на v1.1 автоматически.*

Не путать с 10-migration-v1 — там историческая миграция v0 → v1.0; здесь — правки одной minor-волны с сохранением идентификаторов и трассируемости.

1. Что изменилось — карта волны

#	Правка	Где	Основание	Класс для внедрения
1	**Комплект описания** — единица утверждения, версии и верификации; BR / SR / SPEC / TC-норма без `status` / `version`; `set-version N.M`, линейная история, запись версии	§10.5, §6.5.4, §6.6.4, §8.8, §9.9	ADR-024, #46, #48	**ломающая** (схема)
2	QG-0 / QG-1 / QG-2 — объекты: версия комплекта / реализация TC / версия на версии продукта; `automation.set-version`, `last-run.set-version`	§10.3, §9.3	ADR-024	**ломающая** (схема)
3	§13.3.5 — покрытие парами TC проверяется при QG-0 версии комплекта (`coverage-presence`)	§13.3.5, §10.11.1	ADR-024, ADR-025	обязательное положение — переоценка
4	Манифест: `set-version`, `confirmation: first-party | second-party`	§13.4.2	ADR-024, ADR-017	**ломающая** (манифест)
5	**SPEC-UC** — двенадцатый тип (`role: human | agent`, `steps[].ref`); ссылка шага на утверждение в SPEC-UI / SPEC-PROC; `ux`-TC на действие через интерфейс	§8.3, §8.5.12, §9.8	ADR-018, #35, #34	закрытый список №7; новые правила
6	**MW** — запись ручного прохода (класс свидетельств)	§9.20, reference/02 §8.3	ADR-018, #34, #50	добавляющая
7	**Подтверждение первой стороной** — концепт на входе, ACTZ / AT беспредметны; совмещение ролей	§1.4.4, §5.5.5	ADR-017, #33	добавляющая (вид применимости)
8	**Переиспользуемый компонент** — система в собственном дереве;	§6.14, §10.11.1	ADR-016, #30	добавляющая

	`assumptions[]`, `applies-to`, ребро `uses[]`			
9	`implements[]` — обязательное положение §13.3.8	§13.3.8	#30	обязательное положение — переоценка
10	**Реестр экранов** — перечень в теле SPEC-ARCH, `screens[]` в SPEC-UI, правило покрытия; определение экрана	§8.5.1, §8.5.6.1, §10.7.2	ADR-023, #42, #45, #54	добавляющая (структурная полнота)
11	Полнота охвата обязательного тела SPEC	§8.4.1	ADR-023, #42	добавляющая
12	SPEC-ARCH и SPEC-SEC обязательны и непусты на уровне `system`	§8.3, §10.7.3	модель 3.3	добавляющая (структурная полнота)
13	Контролируемая форма утверждения SR — шесть форм (рекомендация)	§6.6.3.1	ADR-022, ADR-026, #37	рекомендация
14	**Язык описания** — глава 15: операторы, глоссарная дисциплина, запрещённые конструкции, атомарность; обязательна с `RENAR-2`	глава 15, §11.5, §1.7.5 строка 17	ADR-026, #51	обязательна с RENAR-2
15	`automation.status` — два значения (`automated` / `manual-pending` с дедлайном и причиной)	§9.3	#50	**ломающая** (схема)
16	AR фиксирует модель основного агента (`primary.*`); состав `ai-provenance` по канону	§4.10.1, §12.3	#38, #44	схема (ai-provenance)
17	Калибровка порогов метрик — по условию достаточности данных	§12.3	ADR-020	добавляющая
18	Измеритель невырожденности нового обязательного контроля	§13.9.4	ADR-021, #40	процедурная
21	Адрес утверждения `<id>#n`: `verifies[].statement` в TC (обязателен при более чем одном утверждении артефакта), `ref` шага в форме адреса, `coverage-presence` по адресам	§15.1.1, §9.3, §8.5.12.1	ревью v1.1	добавляющая (схема TC)
20	Норма и процесс: стандарт нормирует артефакты, жизненные циклы и правила, не порядок работ; агентские редакции — предусловия вместо шагов	§1.2.1, §1.3 п. 7, §7.6.1, §11.11	ADR-027, #55	редакционная (граница нормы)

2. Что делать внедрению — по классам

2.1 Ломающие правки схемы (строки 1, 2, 4, 15)

1. **Снять `status` и `version` с BR / SR / SPEC / TC-норм.** Состояние артефакта выводится из версии комплекта (§6.5.4). Скрипт миграции: удалить поля, зафиксировать первую версию комплекта `1.0` записью версии (§10.5.4) с подписью Архитектора, включив в неё все артефакты, бывшие `approved` +; артефакты в `draft` остаются в черновике комплекта.
2. **TC:** `verifies[].version` / `requirement-version` → удалить; добавить `automation.set-version` и `last-run.set-version` = версия комплекта, на которой реализация создана и прогнана; `automation.status` → `automated` либо `manual-pending` с `manual-pending-until` / `manual-pending-reason`. Проверка: `scripts/validate-schema-examples.js` (правила `verifies-legacy-version`, `invalid-tc-type`).
3. **Манифест:** добавить `set-version` (текущая утверждённая версия комплекта) и `confirmation` (`second-party` при наличии заинтересованной стороны, `first-party` — при концепте на входе, §1.4.4); `renar-version: "1.1"`.
4. **Хуки носителя:** переходы поартефактных состояний заменить точками контроля версии комплекта (§10.11.1): `coverage-presence`, `implements` -edge, `uses` -edge validation; QG-1 — только допуск реализации TC.

2.2 Обязательные положения (строки 3, 9) — переоценка

Заявление v1.0 обязано быть переоценено (§13.7.3): §13.3.5 теперь проверяется на версии комплекта целиком (каждое утверждение версии — пара TC-норм в той же версии), §13.3.8 требует `implements[]` для BR подсистем. Самооценка — reference/08, ключи `mandatory-clauses-confirmed` не переименованы.

2.3 Добавляющие правки (строки 5–8, 10–12, 16–18)

- **SPEC-UC:** сценарии, ранее жившие как user journey SPEC-UI или happy path SPEC-PROC и сшивающие несколько SR / SPEC, — вынести в `SPEC-UC` с `role` и `ref` на каждом шаге; шаги без `ref` — нарушение структурной полноты. Утверждения SPEC-UI о действии через интерфейс — покрыть `ux` -TC.
- **Реестр экранов:** в SPEC-ARCH уровня `system` добавить раздел «Перечень экранов» из описания (не из маршрутов кода); в каждом SPEC-UI — `screens[]`; непокрытый экран блокирует утверждение версии. Система без интерфейса пишет «интерфейса нет».
- **SPEC-ARCH / SPEC-SEC:** при отсутствии — создать; молчание ТЗ о безопасности закрывается собственным SPEC-SEC, при наблюдаемых клиентом последствиях — через ACTZ.
- **Компонент:** библиотеки и платформы, описанные как подсистемы или «четвёртый уровень», — перевести в систему в собственном дереве с `uses[]` у потребителей (§6.14); путь B или C — по структуре организации.
- **Первая сторона:** внедрения без заинтересованной стороны (прежний §1.5.4) при письменном концепте — вид применимости §1.4.4, `confirmation: first-party`; без концепта — вне области применения.

- **Запись ручного прохода (MW):** ручные прогоны сценариев оформлять записью `MW-NN`; постоянный ручной прогон `ux`-TC не вводится.
- **ai-provenance / AR:** привести `generated-by` к формату с версией, добавить `primary.*` в AR.

2.4 Язык описания (строка 14) — обязателен с RENAR-2

Для заявлений `RENAR-2`+: нормативные разделы артефактов — по §15.2–§15.5. Порядок: (1) операторы — заменить синонимы («обязан», «необходимо», «может») на пять слов списка; (2) термины — свести к одной форме по цепочке, расхождения — находки `terminology`; (3) конструкции §15.4 — переформулировать; (4) атомарность — разбить утверждения, на которые нельзя написать ровно одну пару TC. Формы §15.6 — рекомендация; утверждённые версии задним числом не переформулируются — правки входят следующей минорной версией комплекта.

Строка 20 действий не требует: если ваш производственный процесс отличается от сценариев руководств, он соответствует стандарту, пока выполнены предусловия артефактов (агентская редакция, раздел 4); собственные артефакты и правила — `declared-stricter`.

3. Порядок работ

Шаг	Что	Проверка
A	Инвентаризация: артефакты со `status` / `version`, TC с `requirement-version`, SPEC-UI без `screens[]`, SPEC-ARCH без перечня, отсутствующие SPEC-ARCH / SPEC-SEC	скрипт по frontmatter
B	Первая версия комплекта `1.0`: запись версии, подпись Архитектора; снятие полей	`validate-schema-examples`, структурная полнота §10.7.2
C	TC: `set-version`, `automation.status` из двух значений; QG-1 как допуск реализации	прогон, `last-run.set-version = 1.0`
D	Добавляющие правки §2.3 — следующей минорной версией комплекта `1.1`	QG-0 версии: `coverage-presence`, `screens`, `uses`
E	Язык описания (RENAR-2+) — по мере правки артефактов	состязательный обзор, статическая проверка
F	Манифест `renar-version: "1.1"`, `set-version`, `confirmation`; самооценка reference/08; переоценка заявления	§13.7.3

4. Частые ловушки

- **Версия комплекта ≠ версия продукта.** `set-version` — версия описания; версия продукта живёт в записи верификации (§10.5.4).
- **Минор при всяком изменении.** Правило «крупное или мелкое» не принимается: любое утверждённое изменение — минор; мажор — решение на рубеже с полным аудитом.

- **Реестр из кода.** Перечень экранов по маршрутам реализации нарушает §2.3.1; экран в коде без записи — дрейф и обратная находка.
- **Прописные операторы.** В русской редакции артефактов запрещены (reference/06 §2.1 п. 4); однозначность даёт закрытость списка.

5. Откат

Правки v1.1 не удаляют данных: снятые поля `status` / `version` восстановимы из истории носителя (V1); запись версии комплекта — добавляющая. Возврат к заявлению v1.0 — откат манифеста и повторная самооценка по v1.0.

6. Связанные документы

- ADR-024 (пакетная модель; §9 — места правок по задачам волны) и ADR-026 (язык описания) — архив решений стандарта, доступен в репозитории
- CHANGELOG — запись v1.1
- reference/08 — самооценка соответствия

Guide RENAR 1.1 — renar.tech