RENAR

Стандарт + Руководство + Справочник + авторские мета-документы (полный архив)

RENAR · Полный архив · Версия 1.0-draft | 31.05.2026

Авторы: Вадим Соглаев, Андрей Юмашев

RENAR Core

Версия: 1.3-draft · **Дата:** 2026-06-05 · **Сайт:** renar.tech

> ***Что это.*** *Концептуальный обзор RENAR для человека-читателя: о чём стандарт, зачем нужен, как работает на верхнем уровне. Без технических подробностей, frontmatter, жизненного цикла и нормативных правил — это область полного RENAR Standard.*
>
> ***Время чтения:*** *≤ 10 минут.* ***Для кого:*** *PM, юристы, regulators, инженеры, впервые сталкивающиеся с RENAR. Если вы AI-агент — читайте напрямую Standard, Core вам не нужен.*

Что такое RENAR

RENAR (*Requirements Engineering & Normative Adaptive Regulation*) — нормативный стандарт инженерии требований для разработки с AI-агентами. Стандарт нормирует:

- **Модель данных** артефактов требований: BR (бизнес-требование), SR (системное требование), TR (задача), ADAPT (двусторонняя адаптация ТЗ), 9 типов SPEC (архитектура, API, данные, интеграция, процесс, UI, AI, безопасность, операции), TC (контрольные примеры).
- **Жизненный цикл и контрольные точки качества** (Quality Gates QG-0..QG-4) — состояния артефактов и условия переходов.
- **Возможности носителя** V1–V6, которым должна удовлетворять система хранения артефактов: неизменяемая история, атомарные изменения, сравнение/рецензирование, ветвление, сквозная фиксация версий, автор + отметка времени.
- **Соответствие** — уровни RENAR-1..RENAR-5, обязательные положения, манифест, процедуры оценки.

RENAR — **специализация SENAR** (методологическая база разработки с AI-агентами) в области инженерии требований. Реализация, соответствующая RENAR, всегда совместима с SENAR; обратное — не обязательно.

Зачем существует RENAR

В разработке с AI-агентами требования живут одновременно в нескольких артефактах: договорное ТЗ клиента, инженерные BR/SR/SPEC, тест-кейсы, описание задачи, реализация в коде. Всё это пишет и правит смесь людей и AI-агентов. Без формальных контрактов между артефактами возникает **дрейф требований** — расхождение между тем, что зафиксировано, что верифицируется, и что фактически реализовано.

RENAR закрывает восемь нормированных классов дрейфа:

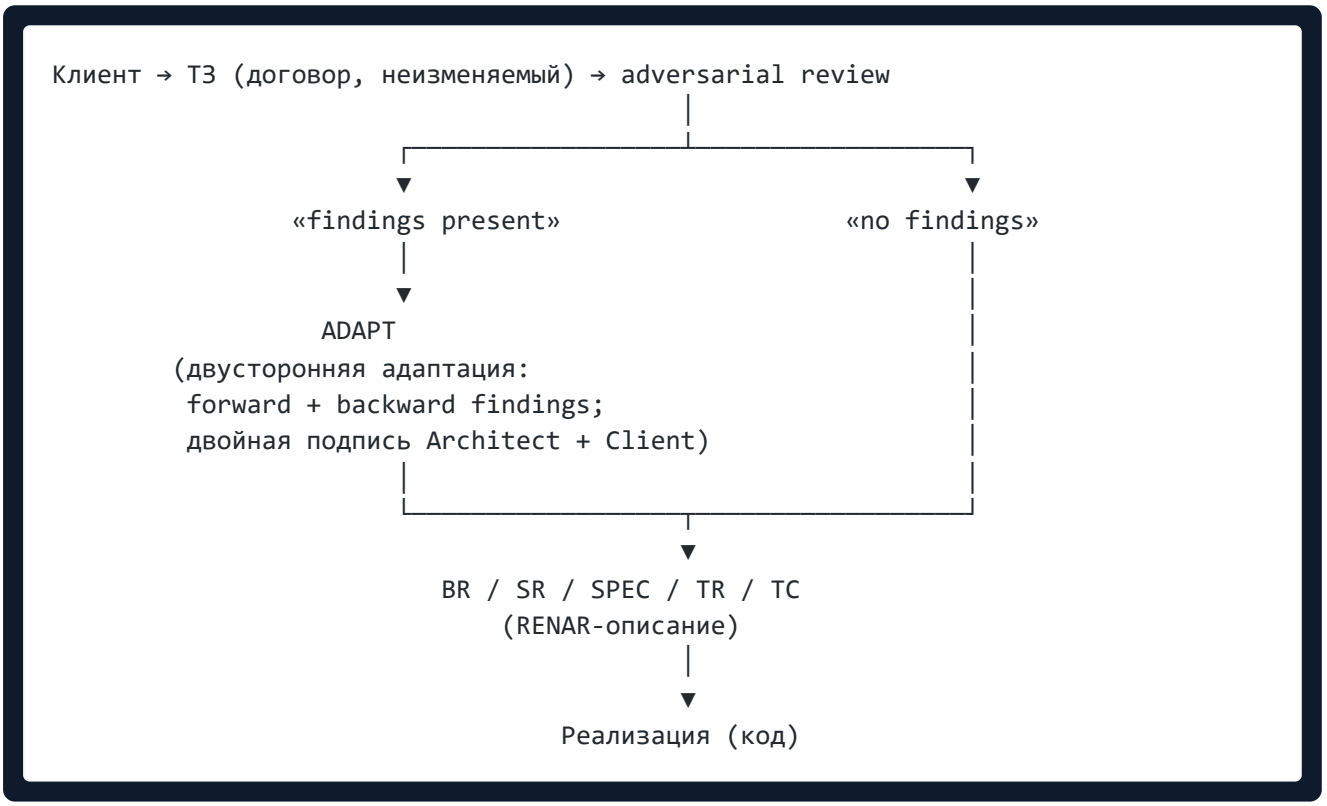
1. **Схемный дрейф** (schema drift) — поля артефактов расходятся между проектами.
2. **Дрейф жизненного цикла** (lifecycle drift) — статусы (`draft` / `approved` / `verified`) значат разное у разных авторов.
3. **Source-of-truth drift** — одна сущность правится одновременно в нескольких местах.
4. **Implementation drift** — код реализует требование, которое уже удалено или переименовано.

5. **Terminological drift** — термины значат разное у разных людей.
6. **Order / provenance drift** — delta-T3 применяется не в порядке, ссылается на несуществующее требование.
7. **TC ↔ requirement provenance drift** — тест верифицирует устаревшее поведение.
8. **Test-fitting drift** — AI-агент ослабляет критерии теста вместо исправления кода.

Все восемь классов — **структурные**: они возникают из самого факта совместного владения артефактом несколькими авторами, а не из ошибок дисциплины. Закрыть их можно только нормативно — фиксацией контракта о том, как артефакты связаны, кто пишет какое поле, и какие предусловия обязаны выполняться при переходах состояний.

Как работает RENAR (концептуально)

Полный жизненный цикл одного требования:



Ключевые свойства:

- **T3 — договорной неизменяемый артефакт.** После подписания клиентом не редактируется. Изменения scope формализуются через delta-T3.
- **ADAPT — реактивный мост между языком клиента и языком требований.** Создаётся только когда конвертация T3 → RENAR требует согласования с клиентом (выявленные backward findings, term mapping, scope clarification). Состязательный рецензент (отдельный AI-агент с другой моделью) фиксирует вердикт «findings present» либо «no findings» для каждого T3.
- **RENAR-описание — источник истины (Source of Truth) о поведении системы.** Код — производный артефакт реализации, не авторитетное определение поведения. Если код делает X, а SR говорит Y — это дефект кода, не «фактическое требование изменилось».
- **TC pos/neg парность.** Каждое верифицируемое утверждение требования имеет минимум один позитивный и один негативный тест-кейс. AI-агенты охотно покрывают happy path и

обходят негативные сценарии — RENAR делает парность нормативной.
- **Состязательный обзор.** Отдельный AI-агент с другой моделью специально ищет, что primary агент пропустил: недостающие backward findings, ослабленные критерии ТС, скрытые допущения. Это компенсирующий механизм против самосогласованных, но семантически неверных AI-выводов.
- **Двойная подпись ADAPT.** Когда ADAPT создаётся, он переходит в approved только после двух подписей: клиент (или его представитель) подтверждает совпадение интерпретации с намерением; архитектор подтверждает техническую реализуемость и закрытие всех находок.

Полный жизненный цикл нормирован через **Quality Gates** (QG-0 Approval, QG-1 Implementation, QG-2 Verification обязательные; QG-3 Architecture, QG-4 Acceptance опциональные). Каждый переход в более высокий статус артефакта проходит через соответствующий gate с зафиксированным участником и предусловиями.

Штатный исполнитель — AI-агент

RENAR-артефакты **штатно** создаются и поддерживаются AI-агентом по заданию инженера. Человек выполняет роль verifier и approver: просматривает результат, уточняет задачу при необходимости, утверждает переходы жизненного цикла.

Из этого позиционирования следуют две вещи, которые непривычны при чтении стандарта в первый раз:

- **Артефакты выглядят плотно** (десятки полей frontmatter, переходы жизненного цикла, графовые связи) — потому что основной читатель машинный. Плотность — не бюрократия, а требование к «коду на естественном языке», который AI-агент исполняет на последующих шагах.
- **Процессные издержки на ведение — машинные, не человеческие.** AI-агент не устаёт заполнять frontmatter; объём работы линейный. Для человека эти издержки кажутся неподъёмными — но именно их и не нужно нести вручную.

При этом **человек остаётся источником решений** по contractual outcomes: подпись ADAPT, утверждение QG-0, выборочная проверка тестов, acceptance результата. AI-агент — responsible (исполняет), человек — accountable (отвечает за результат).

Кому пригодится RENAR

RENAR создан для **контракт-ориентированной разработки**: проектов с явным договорным ТЗ и идентифицируемой стороной клиента, перед которой за ТЗ отвечают. Типичные контексты:

- **Заказная разработка** — независимый vendor + клиент с подписанным ТЗ и acceptance criteria.
- **Регулируемые отрасли** (медицина, финансы, госсектор) — где compliance audit обязателен по нормативу.
- **Enterprise консалтинг** — третья сторона реализует по ТЗ корпоративного клиента с утверждением несколькими заинтересованными сторонами.
- **Public-sector / государственный IT** — тендерные ТЗ, формальная приёмка, multi-year contracts.
- **Long-lived продукты** — где Владелец продукта играет роль представителя Клиента для внутренних feature-ТЗ.

RENAR **не применим** для lean startup discovery, pure R&D без определённого scope, hackathon proofs-of-concept и других контекстов без неизменяемого Т3 и идентифицируемой Заинтересованной стороны.

Маршруты по ролям

Роль	Где начать
PM / RTE	guide/05 — RENAR vs SAFe; затем guide/09 §E3 — практический пример
Legal / Compliance	guide/09 §E3 → guide/06 → reference/07 — ISO 29148 mapping
Regulator / Auditor	reference/07 → reference/08 → standard/13 — манифест соответствия
RE-инженер / Архитектор	guide/00 Быстрый старт → standard/06 → standard/10

Где читать дальше

Документ	Назначение
standard/ — 15 нормативных глав	Полное нормативное описание; обязательное чтение для AI-агента и assessor-а
guide/00-quickstart	30-минутный практический сквозной пример: Т3 → ADAPT → SR → SPEC → TC
guide/01-walkthrough	Расширенный пример на полномасштабном сценарии
guide/06-compliance	GDPR, ФЗ-152, AI Act mapping
reference/01-glossary	Канонический глоссарий + mapping на ISO 29148, BABOK, SAFe, NIST AI RMF
reference/02-schemas	Машино-читаемые схемы артефактов (JSON Schema), правила валидации
reference/03-ai-risk-register	14 AI-рисков по ISO/IEC 23894 + NIST AI RMF

00. Введение

> ***Часть RENAR Standard v1.0-draft*** · *← Оглавление*

> ***Новичкам.*** *Эта глава — нормативная и плотная (RFC-2119, закрытые списки, обязательные положения). Если ты человек и впервые встречаешь RENAR — начни с концептуального обзора* `core/renar-core.md` *(≤ 10 мин), затем* `guide/00-quickstart` *(≈ 30 мин), затем возвращайся сюда. Если ты AI-агент — переходи напрямую к нормативным главам.*

0.1 Зачем эта глава

Клиент написал в ТЗ: «пользователь выгружает отчёт». Инженер прочитал это как выгрузку в PDF, спецификация назвала CSV, а тест в итоге проверяет Excel. Злого умысла нет — требование просто живёт сразу в пяти местах (договорное ТЗ клиента, инженерная декомпозиция, спецификации, тест-кейсы, код), и на каждом стыке смысл чуть смещается. Когда артефакты пишут вперемешку человек и AI-агент, стыков больше, а смещение быстрее. Поэтому узкое место разработки — уже не скорость кода, а **корректность требований**. RENAR существует, чтобы на этих стыках смысл не утекал: он задаёт явные контракты между артефактами.

Эта глава отвечает на четыре вопроса: **что** такое RENAR (§0.2), **зачем** он нужен (§0.3), **как** соотносится с SENAR (§0.4) и **каков минимум**, ниже которого о соответствии говорить нельзя — Minimum Viable RENAR (далее MVR; §0.5). Закрытые списки и язык RU-корпуса — §0.6, §0.7.

Глава **не** вводит новых нормативных требований. Каждое из семи утверждений MVR — ссылка на главу §1–§13, где оно превращается в точную норму; здесь дано связное целое.

> ***Конвенция модальных глаголов.*** *Нормативная сила выражается русскими модальными словами: «обязан» / «должен» — обязательный уровень (RFC-2119* ***shall*** */* ***must****); «следует» / «рекомендуется» — recommended (****should****); «может» / «допустимо» — permitted (****may****); «запрещён» / «не должен» —* ***shall not****. Соответствие — RFC-2119 + ISO/IEC/IEEE 29148:2018 §5.2.1. Действует во всех нормативных главах.*

0.2 Что такое RENAR

RENAR (*Requirements Engineering & Normative Adaptive Regulation*) — нормативный стандарт управления требованиями для разработки с AI-агентами. Стандарт нормирует:

- **Модель данных** артефактов требований (BR / SR / TR), ADAPT, девяти типов SPEC, TC.
- **Жизненный цикл** артефактов через закрытый список Quality Gates (QG-0 / QG-1 / QG-2 обязательные; QG-3 / QG-4 опциональные).
- **Возможности носителя** V1–V6.
- **Соответствие** — уровни RENAR-1..RENAR-5, обязательные положения, манифест, процедуры оценки.

RENAR — **специализация SENAR** (§1.6) в области инженерии требований. Реализация, соответствующая RENAR, **всегда** совместима с SENAR; обратное — неверно (§1.6.2).

RENAR не привязан к конкретному носителю (VCS, документная БД, wiki): нормативные главы используют язык, независимый от носителя, нормируют **возможности** V1–V6 (§3.1). RENAR — стандарт, а **не** реализация.

0.2.1 AI-агент как штатный исполнитель

RENAR-артефакты **штатно** создаются и поддерживаются AI-агентом по заданию инженера. Человек выполняет роль проверяющего и утверждающего. Это нормативное позиционирование, а не методологическая рекомендация.

Следствия:

1. **Полнота** — требования к полноте артефактов: AI-агент исполняет «код на естественном языке», без полноты разрушается происхождение (§0.3).
2. **Плотность frontmatter** — десятки полей следствие машинной природы основного читателя. В многоязычном UI поля могут отображаться человекочитаемо (§4.13.3).
3. **Компенсирующие механизмы** — двойная подпись ADAPT (§7.5), выборочная проверка инженера (§9.14), состязательный обзор (§9.4), Quality Gates (§10.3) — обеспечивают то, что человек остаётся источником решений по договорным результатам.

Что **не** утверждает §0.2.1: RENAR не требует AI-агента для соответствия — стандарт реализуем вручную, но процессные издержки делают это непрактичным (§1.4.1 признак 5). AI-агент не заменяет утверждение человеком — все нормативные подписи выполняются человеком. AI-агент действует как responsible (R в RACI §5.6), не как accountable (A).

0.3 Зачем существует RENAR

В разработке с AI-агентами требования живут одновременно в нескольких артефактах, создаваемых и поддерживаемых смесью авторов-людей и AI-агентов. Без нормативных контрактов между ними возникает **дрейф требований** — расхождение между тем, что зафиксировано, верифицируется и реализовано.

RENAR закрывает **восемь нормированных классов дрейфа** (полный список с точками контроля — §4.11; концептуальное описание — core/renar-core.md). Все восемь — **структурные**: возникают из самого факта совместного владения артефактом несколькими авторами, не из ошибок дисциплины. Закрыть их можно только нормативно — фиксацией контракта связей, авторства полей и предусловий переходов (§13.3 обязательные положения).

Контракт обеспечивается машинно только при условии полноты артефактов (§0.2.1). Артефакт с умолчаниями — контракт с дырами, через которые проходит дрейф независимо от наличия hooks: hook видит только то, что записано. То же относится к каноническим именам и закрытым спискам (§4.2): hook сравнивает строки, не интерпретирует синонимы.

0.4 Связь с SENAR

RENAR нормирует **только** те аспекты инженерии требований, которые SENAR оставляет на усмотрение доменного стандарта: модель данных артефактов, жизненный цикл артефактов требований, манифест соответствия для инженерии требований. RENAR **не** дублирует и **не** переопределяет SENAR-конструкции (5 ценностей, 14 правил, общие Quality Gates, 5 базовых ролей).

Манифест соответствия проекта (§13.4) обязан декларировать одновременно `senar-version` и `renar-version`. Заявление о соответствии RENAR без указания совместимой SENAR-version — несоответствующее (§13.8).

Если нормативное утверждение RENAR оказывается несовместимым с нормой SENAR — это баг стандарта RENAR. Разрешение — через формальную процедуру изменения стандарта SENAR с согласующей правкой в RENAR (§13.9), а не через локальное переопределение на уровне проекта.

Аспект	SENAR (база)	RENAR (специализация инженерии требований)
5 ценностей + 14 правил + 5 ролей	нормирует	наследует
Общие Quality Gates	общий фреймворк	закрытый список канонических QG-0..QG-4 (§10.3)
Модель данных артефактов	— (вне области)	BR / SR / TR / ADAPT / 9 SPEC types / TC (§6–§9)
Жизненный цикл артефактов требований	—	канонические машины состояний (§10)
Возможности носителя	—	V1–V6 (§3)
Манифест соответствия	типовой SENAR	RENAR-CONFORMANCE.yaml + обязательные положения (§13)

Оригинальный вклад RENAR (не унаследовано из SENAR): модель данных требований; ADAPT с двусторонней адаптацией и двойной подписью; независимые от носителя V1–V6; канонический жизненный цикл и QG для оси требований; pos/neg-парность и judge ≠ production как блокирующие положения; уровни соответствия RENAR-1..RENAR-5.

0.5 Minimum Viable RENAR (MVR)

Minimum Viable RENAR — закрытый список из семи нормативных утверждений, обязательных для любой RENAR-соответствующей реализации независимо от объявленного уровня (включая `RENAR-1`). MVR эквивалентен обязательным положениям §13.3.

#	Утверждение[1]	Нормативный источник
MVR-1	**Инверсия источника истины**: иерархия артефактов требований обязана быть источником истины о поведении системы; код — производный артефакт. Обратная разработка (reverse-engineering) поведения из кода в SR без обоснования bug-fix запрещена.	§2.3, §13.3.1
MVR-2	**Возможности V1–V6**: носитель проекта обязан удовлетворять всем шести возможностям — immutable history (V1), atomic change unit (V2), diff & review (V3), branching / change-set (V4), cross-substrate version pin (V5), author + timestamp (V6).	§3.3, §13.3.2
MVR-3	**Reactive, stage-agnostic ADAPT (0..N на T3)**: ADAPT — реактивный артефакт, создаётся тогда и только тогда, когда конвертация T3 → RENAR-описание **на любой стадии деривации** порождает gap между языком клиента и языком	§7, §13.3.3

	требований. На одно ТЗ приходится **ноль или более** ADAPT; каждый привязан к своему триггеру (импорт ТЗ либо конкретная стадия декомпозиции) через `trigger-stage`, множественность — штатный случай. При наличии gap ADAPT обязателен в статусе `approved` с двойной подписью. При отсутствии gap (состязательный рецензент вынес вердикт «no findings») ADAPT не создаётся; BR/SR/SPEC ссылаются на ТЗ напрямую через обязательные `source.tz-section` + `source.adversarial-review-ref`. Delta-ТЗ следует тому же правилу.	
MVR-4	**Закрытый список 9 типов SPEC**: тип SPEC обязан принадлежать закрытому списку `SPEC-ARCH / API / DATA / INT / PROC / UI / AI / SEC / OPS`. Проект не имеет права создавать новые типы локально.	§8.3, §13.3.4
MVR-5	**TC pos/neg парность**: каждое нормативное утверждение верифицируемого артефакта, охватываемое хотя бы одним TC, обязано иметь парный negative TC. Исключение — утверждение само описывает negative invariant.	§9.7, §13.3.5
MVR-6	**Закрытый список Quality Gates**: реализация обязана поддерживать QG-0 (Approval), QG-1 (Implementation), QG-2 (Verification) как `required`. QG-3 / QG-4 — `declared` или `absent`. Новые gate-типы локально создавать запрещено.	§10.3, §13.3.6
MVR-7	**Манифест соответствия**: проект обязан содержать манифест с одновременным `renar-version` + `senar-version` + `level` (RENAR-1..RENAR-5) + подтверждением обязательных положений §13.3. Манифест immutable (V1).	§13.4

[1] Каждая строка задаёт обязательный уровень требования (RFC-2119 «shall» в русских формулировках — «обязан» или «должен» по контексту) в трактовке ISO/IEC/IEEE 29148:2018 §5.2.1.

Реализация, удовлетворяющая всем семи MVR, соответствует минимум уровню `RENAR-1` (§13.2.1). Реализация, нарушающая хотя бы одно MVR, **не имеет** соответствия RENAR независимо от заявленного уровня (§13.8).

0.6 Политика закрытого списка

Список из семи MVR — закрытый на v1.0. Проект не имеет права расширять MVR локально или сокращать список.

Реализация может **ужесточить** требования сверх MVR через маркер `declared-stricter` (§10.10.2, §13.4): требовать QG-3/QG-4 как `required`, требовать состязательный обзор TC на всех типах SPEC, декларировать `RENAR-3+` как минимум.

Реализация **не** имеет права declared-weaker: выдать заявление о соответствии RENAR без поддержки одного из MVR-1..MVR-7; объявить ADAPT опциональным; допустить single-TC-coverage для нормативного утверждения без negative-invariant исключения; опустить `senar-version` или `level` в манифесте.

Изменение MVR — только через формальную процедуру изменения стандарта RENAR (§13.9): исследовательский черновик → публичное обсуждение → повышение minor-версии → руководство по миграции.

0.7 Язык RU-корпуса

RU нормативный корпус RENAR — гибрид: **канонические идентификаторы** (латиница и аббревиатуры закрытого списка) + **русская связующая проза**. Политика — reference/06 §1 и §4.2. Артефакты и ID, статусы жизненного цикла, VCS domain terms, accepted technical loanwords остаются латиницей; организационная терминология и rewrite-кальки — переводятся; editorial rule «≤ 1 latin token на нормативное предложение» вне канонических идентификаторов.

Носитель, не `substrate`. Концепт хранилища по-русски — **«носитель»** (склоняется). Латиницей `substrate` остаётся только в именах полей (`substrate-capabilities`), коде/YAML и именах файлов; в связующей прозе — всегда «носитель».

Оглавление · **Следующая: 01. Область применения →**

01. Область применения

> ***Часть RENAR Standard v1.0-draft*** · *← Оглавление*

1.1 Где RENAR работает, а где нет

Две команды пишут код с AI-агентами. Первая делает банковский модуль по подписанному ТЗ: есть заказчик, приёмка, аудит — каждое требование надо уметь предъявить. Вторая на стартап-скорости проверяет гипотезы: «требование» сегодня есть, а после разворота завтра его нет. RENAR — для первой. Он создан там, где есть договорное ТЗ и сторона, перед которой за него отвечают: заказная разработка, регулируемые отрасли, консалтинг по чужому ТЗ. На чистом продуктовом дискавери — «сначала строим, потом понимаем, что строили» — RENAR не просто избыточен, он структурно неприменим: нет неизменяемого ТЗ, не из чего строить ADAPT. Эта глава проводит обе границы — **предметную** (что стандарт нормирует) и **контекстную** (когда им вообще стоит пользоваться) — закрытыми списками §1.2–§1.5.

Содержательные нормы артефактов и жизненного цикла глава **не** задаёт — это область глав 06–14; нативные для носителя механизмы реализации — область главы 3 и `guide/03-tool-guide-*.md`.

1.2 Что нормирует RENAR (закрытый список)

Закрытый список нормативных областей RENAR на версии v1.0. Каждая область рассматривается в указанной главе:

#	Область	Глава
1	Иерархия требований (BR / SR / TR)	06
2	ADAPT (двусторонняя адаптация ТЗ): forward интерпретация + backward findings + двойная подпись	07
3	Закрытый список 9 типов спецификаций (SPEC-ARCH / API / DATA / INT / PROC / UI / AI / SEC / OPS)	08
4	Test Cases как полноценный артефакт (TC); pos/neg парность; spec-specific TC-обязательства	09
5	Состояния жизненного цикла + Quality Gates (QG-0 / QG-1 / QG-2 обязательные; QG-3 / QG-4 опциональные)	10
6	независимые от носителя возможности версионирования V1–V6	03
7	Maturity model (RENAR-1..RENAR-5)	11
8	Метрики инженерии требований (RDLT, Hallucination Rate, DRA, ACR и др.)	12
9	Соответствие (обязательные положения, манифест, self-assessment, third-party assessment)	13

10	Роли и ответственность за артефакты (specializations над SENAR §4)	05

Все перечисленные области имеют нормативное содержание: обязательные требования и запрет локальных для проекта расширений по соответствующим закрытым спискам.

1.2.1 Что относится к нормативной области

К нормативной области RENAR относятся:

- **Модель данных артефактов** — обязательные frontmatter поля, типы, enum-значения, инварианты согласованности.
- **Состояния жизненного цикла + переходы** — машины состояний per артефакт-тип; pre/post-conditions; gate-id для каждого перехода.
- **Идентичность и происхождение** — неизменяемые идентификаторы; нативная для носителя фиксация авторства и времени (V6); version pin между артефактами (V5).
- **Cross-artifact constraints** — `verifies[]`, `constrained-by[]`, `parent`, `delta-of`, `verified-by`, `implements-spec` — нормативная семантика связей и правила консистентности.
- **AI-происхождение** — обязательная фиксация модели-генератора, prompt-template, объёма tokens; правила human-edits для утверждения.
- **Процедуры соответствия** — обязательные положения, манифест-схема, периодичность оценки, обработка потери соответствия.

1.3 Что RENAR явно НЕ нормирует (закрытый список)

Закрытый список областей, которые **намеренно** оставлены за пределами стандарта. Реализация в этих областях — свободный выбор и не влияет на соответствие.

#	Область	Что вне области
1	**SENAR-методология в целом**	5 ценностей, 14 правил, общие Quality Gates, agent instrumentation — нормируются SENAR; RENAR не дублирует и не переопределяет.
2	**Конкретный носитель / VCS**	Выбор конкретного version-control / document-database / wiki-platform — на усмотрение реализации; RENAR нормирует только возможности V1–V6 (§3).
3	**Tech stack реализации**	Языки программирования, фреймворки, базы данных, инфраструктурные компоненты — вне области; RENAR нормирует требования, не реализацию.
4	**Конкретный UI / IDE / редактор артефактов**	Web-интерфейсы, IDE-расширения, CLI-инструменты — вне области; нормируется только формат хранения артефактов в носителе.
5	**Конкретные test runners**	pytest, jest, playwright, ragas, и др. — специфичны для носителя; RENAR нормирует только обязательность `automation.location` и `last-run` фиксации (V5+V6).
6	**Бизнес-процессы продаж / договоров**	Pre-sale, формулировка договорной стоимости, юридическая структура контрактов — вне области; RENAR работает с ТЗ как с **входом** и не нормирует процесс его создания на стороне клиента.

7	**Project management practices**	Agile-ceremonies, sprint planning, kanban-boards, story-points estimation — вне области; RENAR нормирует workflow артефактов требований, не management-overhead.
8	**AI model selection и prompt engineering**	Выбор LLM-провайдера, prompt-templates, fine-tuning стратегии — вне области; RENAR нормирует только обязательную фиксацию `ai-provenance.generated-by` и состязательный обзор (§9.4).
9	**Конкретные нативные для носителя команды и hooks**	специфичные для носителя CLI-команды (например, конкретные имена команд VCS), реализация hooks (§10.11) — специфичны для носителя и относятся к `guide/03-tool-guide-*.md`.
10	**Юридическая интерпретация artifact-подписей**	Электронная подпись, юридическая значимость, GDPR-обработка персональных данных в ТЗ — вне области стандарта; нормируется применимым законодательством.

1.3.1 Принцип независимости от носителя

Нормативные главы стандарта RENAR используют независимую от носителя терминологию (§3.1). Зависящие от носителя имена (конкретные продукты, протоколы, команды) не появляются в нормативном тексте; они присутствуют только в `guide/` (специфичные для носителя инструменты) и `reference/` (примеры).

1.4 Область применимости (primary scope)

1.4.1 Контракт-ориентированная разработка

Нормативный primary-контекст RENAR — **контракт-ориентированная разработка**: проект, в котором:

1. **Существует ТЗ** (явно сформулированное требование со стороны заказчика), которое после подписания становится **неизменяемым** (§7.4.1 ADAPT-source).
2. **Имеется идентифицируемая сторона клиента** (заинтересованная сторона с полномочиями подписания, §5.3.6) — способная подтвердить acceptance (§10.4.2) и подписать ADAPT (§5.5).
3. **Реактивная двусторонняя client-side validation** — состязательный обзор ТЗ обязателен; если обнаружены findings или требуется уточнение, forward интерпретация ТЗ обсуждается с клиентом через ADAPT (§7.3, §7.4.1). Если конвертация однозначна — validation сводится к зафиксированному вердикту «no findings» (без взаимодействия с клиентом).
4. **Delta-ТЗ workflow** возможен — изменения scope формализуются через delta-ADAPT (§7.6), а не через скрытую переинтерпретацию.
5. **AI-агент доступен как primary author** артефактов RENAR (§0.2.1). Без AI-агента стандарт остаётся применимым, но процессные издержки на ручное ведение артефактов с полным frontmatter, переходами жизненного цикла и графовыми связями делают соответствие RENAR непрактичным; команда либо принимает эти издержки, либо declared-stricter limit scope (§1.7.2) на критическое подмножество требований.

1.4.2 Типичные представители контракт-ориентированной разработки

Контекст	Характеристики
Заказная разработка по ТЗ	Независимый vendor + идентифицируемый client; формальный договор; acceptance criteria.
Regulated industries	Compliance audit обязателен (медицина, финансы, госсектор); traceability requirements → tests → код требуется по нормативу.
Enterprise консалтинг	Third-party реализует по ТЗ клиента-корпорации; утверждение несколькими заинтересованными сторонами; журнал аудита.
Long-lived product с явным владельцем продукта	Владелец продукта играет роль представителя клиента для внутренних feature-ТЗ; внутренние SLA + audit обязательны.
Public-sector / government IT	Тендерные ТЗ; формальная приёмка; multi-year contracts.

1.4.3 Spec-Driven Development (SDD) — современное имя

Контракт-ориентированная разработка с AI-ускорением — это форма **Spec-Driven Development** (§2.3.4). RENAR — нормативный стандарт для AI-native SDD; не альтернатива SDD, а его специализация в области управления требованиями.

1.5 Где RENAR неприменим (негативный scope)

RENAR нормативно неприменим в контекстах, где структурно отсутствуют предусловия §1.4.1. Заявление о соответствии RENAR (§13.4) для проектов в этих контекстах — несоответствующее (§13.8).

1.5.1 Lean startup / pure discovery

Продуктовая команда строит MVP в условиях неопределённости рынка; «требования» — гипотезы, валидируемые на пользователях, а не неизменяемые договорённости. Вне scope: отсутствует неизменяемое ТЗ (гипотезы перепроверяются после pivot); представитель клиента структурно отсутствует (внутренний продакт = автор + единоличный оценщик — нарушение §5.5.3); delta-ТЗ workflow не применим (pivot = смена scope **целиком**, не дельта). Lean startup команды могут заимствовать **отдельные практики** RENAR (AI-происхождение, состязательный обзор ТС) без заявления о соответствии — допустимо как источник идей.

1.5.2 Pure R&D / исследовательские проекты

Научно-исследовательский проект без определённого результирующего scope (exploratory ML research, novel-algorithm prototyping без заказчика). Вне scope: ТЗ как неизменяемый артефакт отсутствует; acceptance criteria (QG-4 §10.4.2) не формализуемы — критерий «успеха» научного исследования не подписывается заранее; двойная подпись ADAPT неприменима.

1.5.3 Exploratory hackathon / proof-of-concept

Time-boxed exploratory работа без обязательной приёмки клиентом. Те же причины, что §1.5.1 + явный отказ от формальной приёмки.

1.5.4 Internal product без external client

Внутренний инструмент команды; «клиент» совпадает с автором; нет независимой заинтересованной стороны для двойной подписи ADAPT. При отсутствии независимого представителя клиента двойная подпись ADAPT (§5.5) структурно невозможна — `client-signature.signed-by == architect-signature.signed-by` нарушает §5.5.3. Это **тот же базовый дефект**, что в §1.5.1: структурное отсутствие двусторонности.

Реализация может **локально** применять подмножество практик RENAR (неизменяемые идентификаторы, состояния жизненного цикла, V1–V6 для собственной дисциплины), но не имеет права заявлять соответствие RENAR-N. Манифест либо не существует, либо явно декларирует «несоответствие».

Негативный сценарий: попытка объявить RENAR-N для internal product без независимого представителя клиента — несоответствующий (§13.8). Если внутренний продукт приобретает идентифицируемую заинтересованную сторону (внутренний Владелец продукта с полномочиями приёмки) — сценарий выходит из §1.5 в §1.4.2 «Long-lived product с явным владельцем продукта», применимо полное соответствие.

1.5.5 Негативные сценарии — конкретизация

Сценарий	Почему несоответствующий
Проект без письменного ТЗ; требования — устные обсуждения	Нарушение §1.4.1 (1): ТЗ как неизменяемый артефакт отсутствует; ADAPT не имеет source (§7.4.1).
Проект с author == client (одно лицо подписывает обе стороны)	Нарушение §5.5.3 о двух независимых лицах в подписях ADAPT.
Проект, в котором scope пересматривается без формальной фиксации delta-ТЗ	Нарушение §7.6 delta-ADAPT workflow; нарушение §13.3 обязательное положение «ADAPT для каждого ТЗ».
Манифест объявляет tech-stack-specific требования (например, «обязательно использовать Python»)	Нарушение §1.3 (3): tech stack вне scope стандарта; манифест несоответствующий.

1.6 Связь с SENAR

1.6.1 RENAR как специализация SENAR

SENAR — **методологическая база**: 5 ценностей AI-native разработки, 14 правил, агенты-инструкции, общие Quality Gates (§10.2.3), 5 базовых ролей (§5.2).

RENAR — **специализация SENAR в области инженерии требований**: нормирует только те аспекты, которые SENAR оставляет на усмотрение доменного стандарта (модель данных артефактов, жизненный цикл, манифест соответствия). RENAR не дублирует SENAR и не переопределяет SENAR-конструкции.

1.6.2 Совместимость

Соответствующая RENAR реализация **всегда** является SENAR-совместимой. Обратное — не обязательно: SENAR-совместимый проект может **не** заявлять соответствие RENAR, если работает вне primary scope §1.4.

Несовместимость RENAR с SENAR в любой нормативной точке — баг стандарта RENAR; разрешается через формальную процедуру изменения стандарта SENAR с согласующей правкой в RENAR.

1.6.3 RENAR не альтернатива SENAR

Реализация **не** имеет права заявить «следуем RENAR вместо SENAR» — это нарушение §1.6.1. RENAR используется поверх SENAR; манифест соответствия проекта объявляет одновременно SENAR-version и RENAR-version (§13.4).

1.7 Политика закрытого списка (закрытый список)

1.7.1 Что закрыто на v1

Списки §1.2 (нормативные области), §1.3 (исключения), §1.4 (primary scope), §1.5 (negative scope) — закрытые. Локальные для проекта расширения и попытки нормировать через манифест исключённые области (§1.3 (3) tech stack, §1.3 (7) PM practices) — несоответствующий. Добавление новых primary-контекстов или перевод сценария из §1.5 в §1.4 — только через формальную процедуру изменения стандарта.

1.7.2 Declared-stricter допустимо

Реализация может **ужесточить** scope относительно нормативного минимума, явно декларируя в манифесте соответствия (§13.4) с маркером `declared-stricter` (§10.10.2): применять RENAR только к подмножеству требований (security-critical SR); требовать дополнительные artifact-типы (threat-models); запретить RENAR без представителя Клиента. Declared-stricter — допустимая локальная политика; соответствие нормативному уровню сохраняется.

1.7.3 Declared-weaker запрещено

Реализация **не** имеет права declared-weaker относительно §1.2 / §1.3 / §1.4: объявить соответствие RENAR для проекта в §1.5 negative scope; применять RENAR без ADAPT (нарушение §13.3.3); нормировать через локальный для проекта манифест исключённые области §1.3.

1.7.4 Путь расширения

Изменение §1.2 / §1.3 / §1.4 / §1.5 — только через формальную процедуру изменения стандарта (§13.9): исследовательский черновик с обоснованием → публичное обсуждение → повышение minor- или major-версии → руководство по миграции для существующих соответствующих проектов.

1.7.5 Master list of closed lists в RENAR

Настоящий параграф — единый индекс всех закрытых списков стандарта RENAR. Каждый перечисленный список нерасширяем локально на уровне проекта; изменения возможны только через формальную процедуру изменения стандарта (§13.9). Канонический источник для каждого списка — в указанной секции; остальные упоминания являются cross-references.

#	Закрытый список	Канонический источник	Cross-refs
1	Нормативные области стандарта (10 пунктов)	§1.2	§1.7.1
2	Исключения из нормативной области (10 пунктов)	§1.3	§1.7.1
3	Primary scope: контексты применимости (4 признака + 5 типичных представителей)	§1.4	§1.7.1
4	Negative scope: контексты неприменимости (5 контекстов + 4 negative scenario)	§1.5	§1.7.1
5	Роли RENAR (specializations над SENAR §4)	§5	§1.2 (10)
6	Категории backward findings ADAPT (7 категорий)	§7.4.4	§13.3.7
7	Типы спецификаций SPEC-* (9 типов: ARCH/API/DATA/INT/PROC/UI/AI/SEC/OPS)	§8.3	§4.4.1, §13.3.4
8	Нормативные принципы TC (закрытый список)	§9.2	§13.3
9	Типы TC (`tc-type` : 6 значений)	§9.5	§4.5.2
10	Состояния жизненного цикла по типам артефактов (BR / SR / SPEC / TC / ADAPT / T3)	§10.5–§10.8	§4.7–§4.10
11	Quality Gates: обязательные {QG-0, QG-1, QG-2}, опциональные {QG-3, QG-4}	§10.3, §10.4	§10.10, §13.3.6
12	независимые от носителя возможности версионирования V1–V6	§3.x	§1.2 (6)
13	Уровни maturity RENAR-1..RENAR-5 (5 уровней)	§11.3	§13.2, §13.9
14	REQ-специфичные метрики (10 метрик)	§12.3	§12.5
15	Drift classes (нарушения требовательной инфраструктуры)	§4.11	§4.2
16	Запрещённые / устаревшие термины (non-canonical)	§4.14	§4.2

Расширение любого закрытого списка проходит ту же формальную процедуру, что и изменение §1.2–§1.5 (§1.7.4, §13.9).

Политики закрытого списка на уровне глав (§10.10, §12.3, §13.9) являются специализациями настоящего §1.7 для соответствующих списков и **не** вводят независимых процедур.

1.8 Перекрёстные ссылки

Источник	Применение
SENAR (полная методология)	Методологическая база RENAR (§1.6.1); RENAR не дублирует SENAR-нормы.
§5 Roles	Владение артефактами и роли — specializations над SENAR §4 (§1.2 (10), §1.6.1).

§2 Methodology positioning	Три фундаментальных утверждения (инверсия источника истины, waterfall-form ≠ classical waterfall, независимое от носителя версионирование) — обоснование scope §1.4.
§7 ADAPT	ADAPT как нормативное требование §1.4.1 (3) двусторонней client-side validation.
§10 Жизненный цикл и QG	Жизненный цикл + Quality Gates — нормативная область §1.2 (5).
§3 Версионирование носителя	возможности V1–V6 — нормативная область §1.2 (6); принцип независимости от носителя §1.3.1.
§13 Соответствие	Обязательные положения, манифест, потеря соответствия — нормативная область §1.2 (9); negative scenarios §1.5.5.
`core/renar-core.md`	Core-mode boundary case для §1.5.4 (internal product без external client).
`guide/02-transition-guide.md`	Practical guidance для проектов, переходящих с lean-стиля на контракт-ориентированную разработку (специфично для носителя).

← Предыдущая: 00. Введение · Оглавление · Следующая: 02. Положение в типологии методологий →

02. Положение в типологии методологий

2.1 Три утверждения, на которых всё держится

Прежде чем описывать артефакты и процессы, RENAR проговаривает три идеи, на которых стоит всё остальное: **источник истины — требования, а не код**; **форма процесса слоистая, но это не классический waterfall**; **версионирование — обязательное свойство носителя, а не привязка к инструменту**. Звучит как общие слова — но убери любую, и остальные главы рассыпаются. Иерархия требований (глава 6) теряет смысл источника истины, ADAPT — роль моста между договорным и инженерным контурами, спецификации — параллельную ось, тест-кейсы — привязку к версии требования, жизненный цикл — атомарность переходов, версионирование носителя — нормативную опору.

Поэтому эта глава — фундамент; §2.3–§2.5 стоит читать подряд. Все три утверждения — **нормативные положения**: каждое является обязательным положением для любого заявления о соответствии RENAR (глава 13).

2.2 Три фундаментальных утверждения

1. **Инверсия источника истины (Source of Truth inversion).** Источник истины о поведении системы — иерархия артефактов требований. Код — производный артефакт реализации. Это и есть разработка от спецификации (Spec-Driven Development, SDD).
2. **Waterfall-форма ≠ классический waterfall.** Процесс RENAR имеет последовательную слоистую форму с гейтами. Стандарт явно отстраивается от четырёх смертельных грехов классического waterfall.
3. **Независимое от носителя версионирование.** Версионирование — обязательное свойство носителя, реализующего RENAR. Конкретный инструмент версионирования взаимозаменяем. Шесть возможностей (V1–V6) нормируют требования к носителю; детали — в главе 3.

Три утверждения логически связаны (см. §2.6).

2.3 Утверждение 1 — Источник истины о поведении: требования, не код (Source of Truth)

2.3.1 Нормативная формулировка

Источник истины о поведении системы — иерархия артефактов требований: ТЗ → ADAPT → BR / SR / SPEC → TR → TC. Код — производный артефакт реализации этой иерархии. При расхождении между кодом и вышестоящим требованием — нормативно побеждает требование.

2.3.2 Распределение ролей

Уровень	Кто источник истины	Кто производный
ТЗ	Договор с клиентом	—
ADAPT	Двусторонняя интерпретация ТЗ	производный от ТЗ
BR / SR / SPEC	Инженерный стандарт системы	производный от ADAPT
TC	Контракт верифицируемого поведения	производный от SR / SPEC
TR (задача)	Акт выдачи работы исполнителю	производный от SR + SPEC
Код	Реализация	производный от всего вышестоящего

2.3.3 Контракт (обязательные следствия)

Стандарт требует следующего поведения от любой реализации, заявляющей соответствие RENAR:

1. **Запрет обратного восстановления поведения из кода в SR (reverse-engineering).** AI-агент или человек, создающий новый SR или дополнение существующего SR, использует в качестве источника ADAPT и существующие SR — но не наблюдаемое поведение реализации. Обратное восстановление допустимо только при создании bug-fix задачи (см. §2.3.4).
2. **Разделение ревью кода и ревью спецификации.** Ревью кода проверяет соответствие изменения коду. Ревью спецификации проверяет соответствие тестов и реализации требованиям. Это два разных гейта с разными артефактами и разными ревьюерами по умолчанию.
3. **Детектирование дрейфа как hook носителя.** При обнаружении в коде ссылки на SR/SPEC, отсутствующий в носителе требований, атомарная единица изменения в реализационном носителе блокируется (см. глава 10 §10.5, глава 3 §3.5).
4. **Запрет молчаливой адаптации SR под код.** Если реализация делает X, а SR требует Y, ровно одно из двух истинно: (a) реализация ошибочна — создаётся bug-fix задача с целью привести код к SR; (b) SR ошибочен — создаётся delta-ADAPT с обоснованием и подписями для изменения SR. Третий вариант («просто обновим SR под код») запрещён.

2.3.4 Положение в индустриальной типологии

Утверждение 1 является конкретной реализацией парадигмы **Spec-Driven Development (SDD)** — индустриального термина, появившегося в 2024–2025 как ответ на ускорение разработки с AI-агентами. SDD признаёт, что когда AI-агенты способны декомпозировать формальные спецификации в код за минуты, **корректность спецификации** становится критическим ограничением, а не корректность кода.

Стандарт / методология или фреймворк	Соответствующее положение	Связь с RENAR §2.3
ISO/IEC/IEEE 29148:2018 §6.4.5	Requirements management mandates traceability from requirements to implementation	RENAR §2.3 — конкретная реализация этого мандата
BABOK Guide v3 §6.5	Verify requirements before they drive solution work	RENAR §2.3 нормирует верификацию как два разных гейта (code/spec)

ISO/IEC 5338:2023	Жизненный цикл ИИ-систем — требования управляют генерацией артефактов AI	RENAR §2.3 вместе с reference/04 AI Style Guide и состязательный обзор (guide/07 §4.5) поддерживают этот мандат
Spec-Driven Development (industry, 2024-2025)	GitHub Spec Kit, Anthropic spec-first agents, Amazon Kiro, Tessl, BMAD-Method	RENAR — формальный стандарт в этой парадигме

Что здесь ново для RENAR. Сама инверсия источника истины — определение парадигмы SDD, а не изобретение RENAR. Вклад RENAR — её **обеспечение соблюдения**: четыре контрактных следствия §2.3.3 (запрет обратного восстановления в SR, разделение ревью кода и спецификации, drift-hook, запрет молчаливой адаптации SR под код), переводящие парадигму из принципа в проверяемые нормативные требования.

2.3.5 Дифференциация от SDD-инструментов

Industry SDD-инструменты и RENAR — в одной парадигме, но в разных слоях:

Ось	Industry SDD-инструменты (Spec Kit, Kiro, Tessl, BMAD)	RENAR
Природа	Эталонная реализация плюс готовый инструментарий с заданным процессом	Формальный нормативный стандарт (capabilities, жизненный цикл, инварианты)
Носитель	привязан к конкретному инструменту / платформе	независим от носителя (V1–V6); VCS / document-oriented store / иной — взаимозаменяемы
Договорный контур	Обычно отсутствует	ADAPT: двусторонняя адаптация ТЗ + двойная подпись (§7)
Соответствие	Не определён	Заявление о соответствии через манифест + обязательные положения (§13)
Верификация	Тесты как практика	pos/neg-парность + judge ≠ production как блокирующие гейты (§9)

RENAR не конкурирует с этими инструментами: industry SDD-инструмент может быть **нативной для носителя реализацией RENAR** без потери переносимости заявления о соответствии (§14.5.2).

2.4 Утверждение 2 — Waterfall-форма, не классический waterfall

2.4.1 Нормативная формулировка

Процесс RENAR имеет последовательную слоистую форму: ТЗ → ADAPT → BR/SR/SPEC → TR → реализация → TC-прогон → accepted. Это waterfall-образная форма. Стандарт явно отстраивается от четырёх смертельных грехов классического waterfall и не должен интерпретироваться как классический waterfall в смысле Royce 1970.

Это утверждение — **позиционное разъяснение** (снятие ложной аналогии с classical waterfall), а не заявление о новизне: после четырёх отстроек §2.4.2 форма совпадает с V-model и ATDD. Новизна

RENAR — в ADAPT, V1–V6 и переводе практик в нормативные положения. Утверждение остаётся обязательным положением (§2.7) — без него ревьюер натягивает на RENAR неподходящий шаблон.

2.4.2 Четыре отстройки от классического waterfall

Классический waterfall (Royce 1970, индустриальная практика 1970–1990)	RENAR
Один большой проход «требования → дизайн → код → тесты» за квартал или год	**Дельта-ТЗ workflow**: каждое изменение — мини-цикл за дни или часы. Та же форма повторяется сотни раз с AI-ускорением. См. глава 7 §7.6 (delta-ADAPT)
Тесты в конце цикла, после реализации	**ТС — полноценный артефакт верификации** (глава 9). Парные pos/neg ТС создаются вместе с SR/SPEC, не после кода. Это ближе к V-model и ATDD, чем к waterfall
Одностороннее «throw spec over the wall»	**ADAPT — двусторонний документ** по построению. Forward (инженерная интерпретация) + backward (вопросы клиенту). Запрет на одностороннюю передачу
Спека написана один раз, потом неприкосновенна; реальность дрейфует	**Непрерывная сверка** (continuous reconciliation) через hooks носителя. Дрейф code ↔ spec детектируется автоматически и попадает в delta-ADAPT. Спека живая

2.4.3 Применимость

RENAR применим в следующих контекстах:

- Контракт-ориентированная разработка (наличие договора с клиентом, подписанного ТЗ).
- Регулируемые отрасли: соответствие нормам, медицина, финтех, госсектор, обработка PII.
- Корпоративный консалтинг (третья сторона делает продукт по чужому ТЗ).
- Проекты с высокой стоимостью изменений требований поздно в цикле.
- Проекты, требующие журнала аудита для ревизий соответствия (глава 13).

RENAR **не применим** в следующих контекстах:

- Чистый продуктовый дискавери без договорного контекста (lean startup, продукт-MVP «сначала строим, потом понимаем что строим»).
- Чистое R&D без определённых требований.
- Прототипирование с жизненным циклом меньше срока написания ADAPT.

Применимость документируется как часть процедуры соответствия (глава 13).

2.4.4 Положение в индустриальной типологии

Методология	Связь с RENAR
Classical Waterfall (Royce 1970)	RENAR отстраивается по 4 пунктам §2.4.2
V-model	RENAR ближе к V-model: парные ТС, тесты в начале каждого слоя
Scrum / Kanban	Не противоречит, но RENAR — другая ось (артефактная, не процессная). Scrum sprint может содержать RENAR delta-ADAPT cycle

SAFe Solution Intent	ADAPT — конкретная реализация Solution Intent для контракт-ориентированной разработки. См. guide/05-safe-comparison.md
BABOK Requirements Analysis	RENAR §2.3+§2.4 — формальная реализация BABOK §3+§6 для контекста разработки с AI-агентами

2.5 Утверждение 3 — Независимое от носителя версионирование

2.5.1 Нормативная формулировка

Версионирование — обязательное свойство носителя, реализующего RENAR. Стандарт нормирует шесть возможностей (V1–V6), которые носитель обязан обеспечить. Конкретный инструмент версионирования взаимозаменяем: носитель, удовлетворяющий V1–V6, реализует RENAR независимо от того, является ли он распределённым VCS, централизованным VCS, документ-ориентированной СУБД с разрешением конфликтов или иным механизмом.

2.5.2 Шесть обязательных возможностей (обзорно)

Полная нормативная формулировка V1–V6 — в главе 3. На уровне положения в типологии:

#	Возможность	Что обеспечивает
V1	Неизменяемая история	Любое прошлое состояние артефакта восстановимо
V2	Атомарная единица изменения	«Изменение» — одна транзакция; всё или ничего
V3	Сравнение различий и рецензирование	Человек видит изменение и утверждает или отклоняет до интеграции
V4	Ветвление / набор изменений	Черновики отделимы от утверждённой истины
V5	Сквозная фиксация версии	Реализационный носитель фиксирует версию носителя требований
V6	Автор и отметка времени	Каждое изменение имеет автора и время

2.5.3 Носитель, не удовлетворяющий V1–V6, не реализует RENAR

Невозможность реализации RENAR на носителе без V1–V6 — структурная, не операционная: утверждение 1 (инверсия источника истины) физически не работает без версионирования, потому что:

- Невозможно сказать «реализация собрана против требований по состоянию на дату X» — пропадает происхождение (нарушение V1, V6).
- Невозможно построить delta-ADAPT — нет базовой версии для отсчёта дельты (нарушение V1).
- Невозможно верифицировать TC — поле `verifies[].requirement-version` теряет смысл без стабильного идентификатора версии (нарушение V5).
- Невозможен переход требования `verified → accepted` — гейту не на что опереться (нарушение V1, V5).

- Невозможен журнал аудита «что мы сдавали клиенту по контракту 2025-Q3» (нарушение V1, V6).

Поэтому носитель без V1–V6 (плоский файловый сервер с переименованием файлов как механизмом версии; document store без разрешения конфликтов; иные системы без неизменяемой истории) **не реализует RENAR** независимо от других свойств.

2.5.4 Независимый от носителя нормативный язык

Нормативные параграфы RENAR используют независимый от носителя язык: «атомарная единица изменения», «фиксация версии», «автор и отметка времени», «сравнение различий и рецензирование» — а не названия конкретных примитивов инструментов. Это обеспечивает применимость стандарта к любому будущему носителю, удовлетворяющему V1–V6. Специфичные для носителя детали — в Глава 3 §3.4 (нормативная таблица отображения V1–V6 × носителей), guide/03-tool-guide-git.md (distributed VCS), guide/04-document-store-substrate.md (document-oriented store).

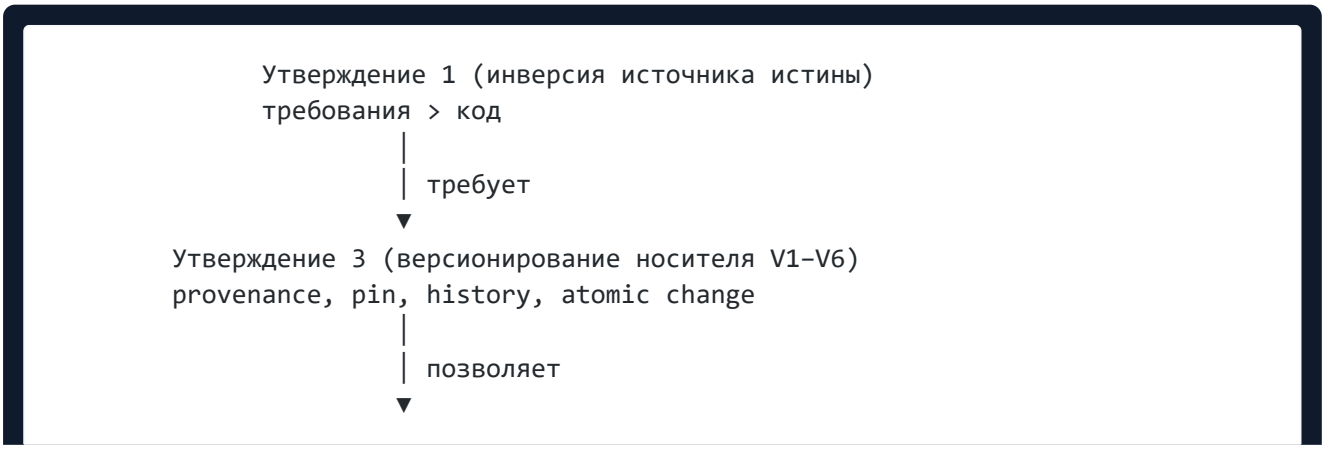
2.5.5 Положение в индустриальной типологии

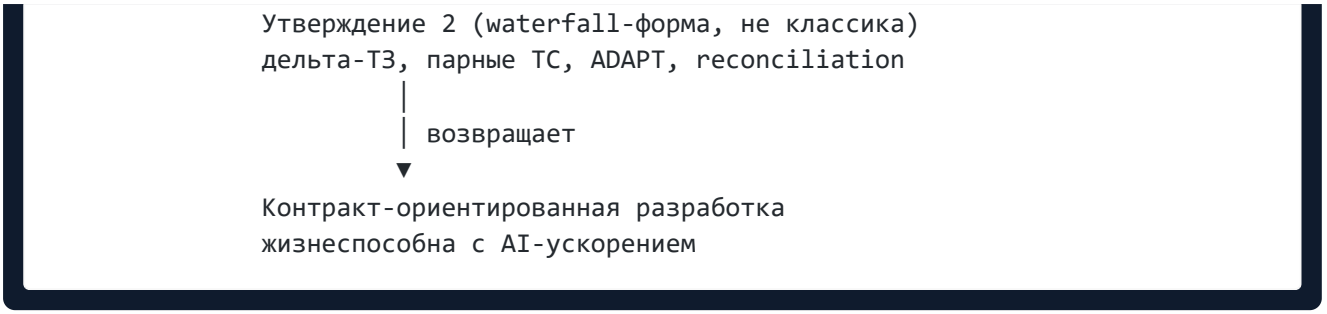
Стандарт	Соответствующее положение	Нейтральность к носителю
ISO/IEC/IEEE 29148:2018 §6.4.5	Configuration management of requirements	нейтрален к носителю; нормирует возможности, не инструменты
BABOK Guide v3 §5.3	Maintain requirements (для прослеживаемости)	нейтрален к носителю
CMMI-DEV CM SG2	Track and Control Changes	нейтрален к носителю
SAFe Solution Intent	Versioned artifact	Не нормирует носитель
ISO/IEC 5338:2023	Происхождение артефактов ИИ (AI artifact provenance)	нейтрален к носителю; происхождение — возможность

RENAR следует той же нейтральности и явно фиксирует V1–V6 как контракт, что эти стандарты оставляли неявным.

2.6 Логическая связь трёх утверждений

Три утверждения связаны направленно:





Без утверждения 1: источник истины диффузен, спека дрейфует, аудит невозможен. **Без утверждения 3**: утверждение 1 декларативно, физически не работает. **Без явного утверждения 2**: ревьюер натягивает на RENAR неподходящие шаблоны (agile sprint без waterfall-form, или classical waterfall без 4 отстроек) и отвергает стандарт по неверной аналогии.

Все три утверждения являются обязательными положениями (глава 13). На уровне v1 заявление о соответствии RENAR требует принятия всех трёх утверждений; без любого из них оно несостоятельно.

2.7 Следствия для процедуры соответствия

Утверждения §2.3, §2.4, §2.5 — **обязательные положения** для любого заявления о соответствии уровням RENAR (RENAR-1..RENAR-5; см. гл.11, гл.13). Команда не может заявить «реализуем RENAR-1 без инверсии источника истины» (противоречие в терминах) или «реализуем RENAR на носителе без V1–V6» (носитель не соответствует стандарту); может выбрать **не заявлять** соответствие RENAR в контексте неприменимости (§2.4.3) — это нормативно допустимо. Конкретный чек-лист самооценки — гл.13.

2.8 Связь с другими главами стандарта

Глава	Связь
06 Требования	Иерархия BR/SR/TR — конкретизация цепь источника истины из §2.3.2
07 ADAPT	Двусторонняя адаптация — конкретизация утверждения 2 (отстройка от «переброса спецификации через забор»)
08 Specifications	SPEC-* как параллельная ось — следствие инверсия источника истины для структурного описания
09 Test cases	TC как полноценный артефакт верификации — конкретизация утверждения 2 (отстройка от «тесты в конце»)
10 Жизненный цикл и QG	Гейты обеспечивают соблюдение утверждения 1 (drift hooks) и утверждения 2 (continuous reconciliation)
03 Версионирование носителя	Детальная нормативка V1–V6 (§2.5 — обзорно)
11 Maturity	Уровни RENAR-1..RENAR-5 расширяют утверждения дополнительными требованиями
13 Соответствие	Самооценка по трём обязательным положениям

03. Версионирование носителя — V1–V6

> ***Часть RENAR Standard v1.0-draft*** · *← Оглавление*

3.1 Шесть возможностей носителя

Инверсия источника истины из главы 2 — побеждает требование, а не код — держится на одном физическом условии: носитель обязан честно помнить, что и когда менялось. Если прошлое состояние можно переписать задним числом, фраза «реализация прошла приёмку против требований версии X» теряет смысл, а вместе с ней рушится весь договор о приёмке. Поэтому шесть возможностей V1–V6 — не инфраструктурная деталь, а фундамент, на котором идея RENAR вообще стоит; их и выносим вперёд, до глав об артефактах.

Глава даёт **детальную нормативную формулировку** каждой возможности: предусловия, постусловия, связь с цепочкой источника истины и примеры отображения на распространённые носители. Концептуальное обоснование — §2.5 (утверждение 3 «Положения в типологии методологий»).

Конкретный механизм версионирования — распределённый или централизованный VCS, document store с разрешением конфликтов — **взаимозаменяем**. RENAR нормирует возможности, а не инструменты.

3.2 Обоснование обязательности

Разберём по одной, что именно ломается без каждой возможности. Связь здесь структурная, а не операционная: дело не в дисциплине команды, а в том, что без возможности соответствующее свойство истины просто нечем выразить.

Нет возможности	Что становится невозможным	Связь в RENAR
V1 (неизменяемая история)	«Реализация собрана против требований по состоянию на дату X»	происхождение, журнал аудита, гейт приёмки
V2 (атомарная единица изменения)	Гарантированная согласованность изменений	delta-ADAPT как атомарная единица изменения (см. §7.6)
V3 (сравнение различий и рецензирование)	Утверждение требований и спецификаций	QG QG-ADAPT-approve, QG-spec-approved
V4 (ветвление / набор изменений)	Черновик отделим от утверждённой правды	переходы `draft → review → approved`
V5 (сквозная фиксация версии)	Привязка версии требования из реализации	поле `verifies[].requirement-version` в TC (§9)
V6 (автор и отметка времени)	происхождение каждого изменения	подписи в ADAPT, `ai-provenance` в SR/SPEC

Поэтому носитель без V1–V6 — плоский файловый сервер без истории, document store без разрешения конфликтов, любой механизм без неизменяемого отслеживания изменений — **не реализует RENAR** независимо от прочих достоинств. Это структурное ограничение, а не вопрос дисциплины команды.

3.3 Нормативные определения V1–V6

Параграфы §3.3.1–§3.3.6 сформулированы **независимо от конкретного носителя**. Имена конкретных продуктов — только в §3.4 (пример) и §3.6 (иллюстрации языка).

3.3.1 V1 — неизменяемая история

Возможность (immutable history): носитель обязан обеспечить, что для любого артефакта любое прошлое состояние адресуемо и восстановимо без потерь.

Предусловия: артефакт зарегистрирован в носителе как объект версионирования.

Постусловия: для любого момента времени T в истории артефакта существует стабильный идентификатор версии, через который состояние на момент T полностью восстанавливается.

Без V1 невозможно:

- Восстановить состояние артефакта на дату подписания контракта.
- Сравнить текущее состояние с опорным.
- Построить журнал аудита для соответствия (глава 13 §13.4).
- Задать базовую точку для delta-ADAPT.

Конкретные реализации на разных носителях — §3.4.

3.3.2 V2 — атомарная единица изменения

Возможность (atomic change unit): носитель обязан обеспечить, что любое изменение артефакта (или согласованной группы артефактов) фиксируется как одна транзакция: всё или ничего. Промежуточные несогласованные состояния снаружи не наблюдаемы.

Предусловия: у носителя есть понятие транзакции (atomic change).

Постусловия: после атомарного изменения либо все правки видны наблюдателю, либо ни одна. Промежуточное «рассогласованное» состояние недопустимо.

Без V2 невозможно:

- Согласованно обновить BR и связанные SR одной транзакцией.
- Гарантировать согласованность требования и метаданных `linked-tasks[]`.
- Оформить утверждение ADAPT как единое атомарное действие (двойная подпись + переход `client-ready → approved` за одну транзакцию).
- Откатить изменение без «полуприменённого» состояния.

3.3.3 V3 — сравнение различий и рецензирование

Возможность (diff & review): носитель обязан обеспечить, что предложенное (но ещё не интегрированное) изменение представимо как diff против опорного состояния, и человек или AI-агент с правом рецензирования может одобрить или отклонить его **до** включения в утверждённую правду.

Предусловия: предложенное изменение существует отдельно от утверждённой правды (см. V4).

Постусловия: до утверждения изменение существует, но не считается частью источника истины. После утверждения становится частью источника истины как атомарная единица изменения (V2).

Без V3 невозможно:

- Quality gates QG-ADAPT-approve, QG-spec-approved, QG-sr-approved (см. глава 10).
- Двойная подпись ADAPT (см. §7.5).
- Независимое рецензирование кода и спецификации (см. §2.3.3 (2)).
- Состязательный обзор как гейт.

3.3.4 V4 — ветвление / набор изменений

Возможность (branching / change-set): носитель обязан разделить **работу в процессе** (WIP) и **утверждённую правду** (источник истины) так, что несколько независимых изменений могут разрабатываться параллельно без влияния на источник истины.

Предусловия: артефакт зарегистрирован в носителе.

Постусловия: для любого артефакта в данный момент могут существовать (a) ровно одна утверждённая версия (источник истины) и (b) ноль или более черновиков — каждый либо интегрируется через V3, либо отклоняется.

Без V4 невозможно:

- Отделить статус жизненного цикла `draft` от `approved` (глава 10).
- Параллельно вести несколько delta-ADAPT.
- Держать backward findings в `asked-to-client` без блокировки утверждённой правды.
- Эволюционировать SPEC-* экспериментально без влияния на требования, производные от реализации.

3.3.5 V5 — сквозная фиксация версии между носителями

Возможность (cross-substrate version pin): носитель A, использующий артефакт носителя B, обязан иметь возможность зафиксировать конкретную версию артефакта носителя B как стабильный сквозной идентификатор. Этот идентификатор должен однозначно восстанавливать состояние артефакта в носителе B.

Предусловия: носитель A и носитель B удовлетворяют V1 (у каждого есть стабильный идентификатор версии).

Постусловия: для каждой зафиксированной ссылки в носителе A на артефакт носителя B существует пара `(artifact-id, version-id)`, и по ней восстанавливается полное состояние артефакта носителя B на момент фиксации.

Без V5 невозможно:

- Поле `verifies[].requirement-version` в TC (глава 9 §9.4).
- Привязать носитель реализации (код) к конкретной версии носителя требований (SR/SPEC).
- Гарантировать: «эта реализация прошла приёмку против требований версии X».
- Метрика свежести TC (зафиксированная версия старше текущей — TC устарел).

3.3.6 V6 — автор и отметка времени

Возможность (author + timestamp): носитель обязан для каждой атомарной единицы изменения (V2) зарегистрировать идентифицируемого автора (человек или AI-агент с уникальным id) и отметку времени с точностью не хуже секунды.

Предусловия: у носителя есть система идентификации авторов.

Постусловия: для любой атомарной единицы изменения запрос «кто? когда?» даёт однозначный ответ.

Без V6 невозможно:

- Двойная подпись ADAPT (подпись = автор + отметка времени в нативной для носителя форме).
- `ai-provenance` во frontmatter артефактов.
- Журнал аудита для соответствия.
- Метрика состязательно-найденного (распределение backward findings по авторам).

3.4 Отображение V1–V6 на конкретные носители (пример)

> ***Информативно.*** *Таблица показывает, как V1–V6 обычно реализуются на распространённых носителях. Она* ***не*** *часть нормативного контракта. Любой носитель, удовлетворяющий V1–V6, реализует главу 3 — независимо от того, есть ли он в таблице.*

Возможность	Git	Mercurial	SVN	Perforce	Document store (пример)
V1 — неизменяемая история	commits, hash-chain	changesets, hash-chain	revisions, sequential numbering	changelists	revision tree per doc (`_rev`)
V2 — атомарная единица изменения	commit	commit	atomic revision	changelist submit	document update (single _rev advance)
V3 — diff и рецензирование	merge request / pull request	hg phabricator, mq	svn diff + commit gate	swarm review	API workflow + Hub UI approval
V4 — ветвление / набор изменений	branches	named branches, bookmarks	branches (copy semantic)	branches / streams	conflict branches / WIP docs / draft status
V5 — сквозная фиксация версии	submodule SHA	subrepo changeset	externals (peg rev)	branches / streams reference	`_rev` reference + `created_by_order`
V6 — автор и отметка времени	commit metadata	commit metadata	revision properties	changelist metadata	doc fields (`author` , `updated_at`)

Практические workflow для каждого носителя:

- guide/03 — git
- guide/04 — document store

3.5 Носитель, не удовлетворяющий V1–V6

Следующие конфигурации **не реализуют RENAR**, потому что нарушают одну или более возможностей:

Конфигурация	Нарушение
Плоский файловый сервер; «версия» = переименование файла	V1 (нет стабильной истории; переименование = потеря происхождения)
Document store без разрешения конфликтов	V2 (возможно рассогласованное состояние)
Wiki без истории ревизий	V1, V6
Wiki с историей ревизий, но без процедуры утверждения	V3
VCS с `mtime` вместо неизменяемого идентификатора	V1, V5
носитель с правкой исторических ревизий на месте	V1 (история изменяема — журнал аудита невозможен)

Команда, **внедряющая** RENAR на носителе из этого списка, **должна** либо перейти на подходящий носитель, либо построить **компенсирующий слой**, обеспечивающий V1–V6 поверх базового хранения. В обоих случаях манифест соответствия обязан явно документировать, как реализованы V1–V6.

3.6 Язык, независимый от конкретного носителя

Нормативные параграфы RENAR используют **независимые от носителя** формулировки: «атомарная единица изменения» (V2), «фиксация версии» (V5), состояние `approved` (после V3), «черновик» (V4). Термины, **специфичные для носителя** (имена инструментов и их примитивы), допустимы только:

- в иллюстративных таблицах с явной пометкой (как §3.4);
- в перекрёстных ссылках на guide/ по конкретному носителю;
- в приложениях к нормативным главам.

3.6.1 Примеры: нормативная форма и иллюстрация на git

Нормативная (независимая от носителя) формулировка	Иллюстрация на git (не норма)
«Изменение требования регистрируется как атомарная единица изменения с автором и временем (V2, V6)»	«Изменение регистрируется как commit»
«Изменение проходит сравнение различий и рецензирование до интеграции (V3)»	«Изменение проходит review merge request до merge в main»
«носитель реализации фиксирует конкретную версию носителя требований (V5)»	«`.src` фиксирует submodule SHA `.req`»

«Двойная подпись ADAPT — два независимых события автор+отметка времени (V6)»	«Двойная подпись — два approve в UI merge request»
«Правка уже утверждённого требования вне атомарной единицы изменения с рецензированием — нарушение»	«Force-push на approved branch блокируется hooks»

3.6.2 Носитель как параметр соответствия

Каждая команда фиксирует **выбор носителя** в манифест соответствия с явным отображением V1–V6 → нативные для носителя примитивы. При смене носителя (например, с git на document store) манифест обновляют и проводят регрессионную проверку V1–V6 (§3.8).

3.7 Манифест соответствия

Каноническая схема — §13.4, файл `RENAR-CONFORMANCE.yaml` (YAML 1.2) в корне носителя требований. Глава 3 нормирует часть, относящуюся к носителю: декларацию выбранного носителя и отображение V1–V6 (поле `substrate-capabilities`, §13.4.2).

```
# Фрагмент RENAR-CONFORMANCE.yaml — substrate-специфичная часть
# (полная схема — §13.4.2)
substrate:
  requirements:   { tool: "<name>", version: "<version>" }
  implementation: { tool: "<name>", version: "<version>" }
v1-v6-mapping:
  v1-immutable-history:   { primitive: "<substrate-specific>", validation: "<CI
check / manual>" }
  v2-atomic-change-unit:  { primitive: "<substrate-specific>", validation: "<CI
check / manual>" }
  v3-diff-review:         { primitive: "<substrate-specific>", validation: "<CI
check / manual>" }
  v4-branching:           { primitive: "<substrate-specific>", validation: "<CI
check / manual>" }
  v5-cross-substrate-pin: { primitive: "<substrate-specific>", validation: "<CI
check / manual>" }
  v6-author-timestamp:    { primitive: "<substrate-specific>", validation: "<CI
check / manual>" }
```

Подтверждение обязательных положений (§2.3 инверсия источника истины, §7 ADAPT, §8 закрытый список 9 типов SPEC, §9 pos/neg и др.) — в поле `mandatory-clauses-confirmed` (§13.3–§13.4).

Манифест **обязателен** для любого заявления о соответствии уровня RENAR-1 и выше (глава 13).

3.8 Миграция носителя

При смене носителя команда **обязана** выполнить шаги по порядку:

1. **Аудит до миграции:** целевой носитель удовлетворяет V1–V6; иначе миграция запрещена до компенсирующего слоя.
2. **Черновик манифеста:** обновлённый `RENAR-CONFORMANCE.yaml` с новым отображением.
3. **Проверка изоморфности:** все артефакты (BR, SR, SPEC, ADAPT, TC) переносимы без потери полей, id и происхождения. Все id **неизменяемы** — переименование при миграции запрещено.
4. **Атомарное переключение:** миграция — атомарная единица изменения на уровне процесса; одномоментный перевод всех артефактов. **Параллельное использование двух носителей как источника истины запрещено** (см. §2.3.3 (1)).
5. **Проверка после миграции:** регрессионная проверка V1–V6 на реальных артефактах.
6. **Регистрация манифеста:** обновлённый `RENAR-CONFORMANCE.yaml` регистрируется в новом носителе как атомарная единица изменения (V2).

После переключения старый носитель — архив (снимок только для чтения) или упразднён. Параллельная работа на двух носителях как источника истины запрещена.

3.9 Связь с другими главами

Глава	Связь
02 Положение в типологии §2.5	Концептуальное обоснование V1–V6
06 Иерархия требований	frontmatter опирается на V1 (неизменяемый id), V5 (фиксация версии)
07 ADAPT	Утверждение двойной подписью — V3 + V6; delta-ADAPT — V1 + V2 + V4
08 Спецификации	`constrained-by[]`, `depends-on[]`, `referenced-by[]` — через V5
09 Тест-кейсы	`verifies[].requirement-version` — прямое применение V5
10 Жизненный цикл и QG	Переходы статусов — V2 + V3 + V4
11 Модель зрелости	RENAR-3+: V5 обязателен; RENAR-4+: V6 + ai-provenance
13 Соответствие	`RENAR-CONFORMANCE.yaml` — обязательный артефакт
guide/03	Практика на git
guide/04	Практика на document store

---

04. Термины и определения

> ***Часть RENAR Standard v1.0-draft*** · *← Оглавление*

4.1 Зачем единый словарь терминов

Одно и то же слово у двух команд часто значит разное: для одной «спека» — это SR, для другой — макет экрана, для третьей — целый API-контракт. Пока термины плавают, рушится прослеживаемость и буксует любой разговор о соответствии. Эта глава убирает разночтения: справочник по формулировкам, который читают при согласовании артефактов, проверке носителя и подготовке оценки соответствия. Она фиксирует одно имя на каждую концепцию и тем гасит терминологический дрейф между командами и инструментами. После оглавления переходите к §4.3–§4.5 по типам артефактов, §4.6–§4.7 для состояний и гейтов, §4.8–§4.10 для носителя и происхождения, §4.11/§4.14 для drift-классов и запрещённых терминов; индекс закрытых списков — §1.7.5.

Глава нормирует **каноническую терминологию RENAR**: одно определение на одну концепцию; единый источник истины для других глав, реализационных носителей, инструментов проверки соответствия. Терминологический дрейф (§4.11) — отдельный класс нарушений соответствия (§13.3.1 косвенно).

Глава **не дублирует** reference/01-glossary.md: эта глава содержит **канонические нормативные** определения короткой формы; `reference/01` — развёрнутые объяснения с антипаттернами, историей и отраслевым контекстом (информационный материал).

4.2 Принцип «только канонический»

RENAR выбирает **один канонический термин** на одну концепцию. Внутри носителя (frontmatter, ID, нормативные абзацы тела, scripts, CI hooks) используется **только канонический термин**. Сопоставление с родственными стандартами (§4.13) — для документации, миграции и интеграции с внешними системами; внутри носителя замены канонического термина на эквивалент из таблицы сопоставления **не происходит**.

При обнаружении неканонического термина (по §4.14) в нормативном артефакте — нативный для носителя hook (§10.11.1) обязан выдать предупреждение при change-set; для RENAR-4+ (§11.7) — блокирующая ошибка.

Многоязычные проекты могут отображать каноническую терминологию в UI на язык клиента (§4.13.3); это **UI-перевод**, не каноническая замена.

4.3 Артефакты требований

4.3.1 ТЗ — Техническое задание

ТЗ (`TZ-YYYY-NNN`) — **неизменяемый** (§7.4.2) договорный артефакт, фиксирующий обязательства между клиентом и инженерной командой. После регистрации в носителе не редактируется.

Изменения — через delta-ТЗ как новый неизменяемый артефакт (§7.6).

4.3.2 ADAPT — мостовой артефакт (bridge artifact)

ADAPT (`ADAPT-NNN`) — обязательный мостовой артефакт (§7.4.1) между ТЗ и иерархией требований. Содержит forward (инженерная интерпретация) + backward (вопросы клиенту) разделы. Каждое ТЗ обязано иметь ровно один корневой ADAPT в статусе `approved` (§13.3.3). Жизненный цикл: §4.6.4.

4.3.3 BR — Business Requirement

BR (`BR-NN`) — артефакт бизнес-уровня. Описывает наблюдаемый бизнес-эффект (`business-outcome`), а не способ его достижения. Декомпозируется в SR. Frontmatter — §6.5.2. Жизненный цикл: §4.6.1.

4.3.4 SR — System Requirement

SR (`SR-NN`) — артефакт системного уровня. Описывает обязательное поведение системы в рамках одного бизнес-эффекта. Имеет родительский BR (`parent: BR-N`) или родительский SR (при `level: subsystem` или `level: module` — §6.7). Frontmatter — §6.6.2. Жизненный цикл: §4.6.1.

4.3.5 TR — Task Requirement

TR (`TR-NN`) — артефакт уровня задачи исполнителя. Описывает практически выполнимую работу с goal + AC. Имеет родительскую цепочку `implements: SR-N` (или BR для тривиальных задач). Frontmatter — §6.7.2. Жизненный цикл: §4.6.2.

4.3.6 Иерархия

Иерархия требований:

```
ТЗ → ADAPT → BR → SR → TR → реализация
                    │
                    └── SPEC-* (параллельная ось, §4.4)
```

SR может реализовывать SPEC через `implements-spec[]` (§8.6.2) — это **связь**, а не parent-ребро.

4.4 SPEC артефакты

SPEC — артефакт структурного описания системы как параллельная ось требований (§8.2). Не parent-ребро с SR; связан через `constrained-by[]` / `implements-spec[]` (§8.6). Жизненный цикл: §4.6.3.

4.4.1 Закрытый список 9 типов SPEC

Закрытый список (§8.3):

Тип	Назначение
`SPEC-ARCH`	Архитектура системы (контексты, контейнеры, компоненты, deployment view, quality attributes)
`SPEC-API`	API contracts (REST / GraphQL / gRPC / async events)
`SPEC-DATA`	Модель данных (схема, ERD, migrations, retention, PII classification)
`SPEC-INT`	Integration (взаимодействие между подсистемами и внешними системами)
`SPEC-PROC`	Process / workflow (бизнес-процессы, машины состояний, saga)
`SPEC-UI`	UI / UX (экраны, навигация, accessibility, baselines)
`SPEC-AI`	AI / ML (model cards, RAG, prompt engineering, eval strategy)
`SPEC-SEC`	Security (authn / authz, threat model, secrets management)
`SPEC-OPS`	Operations (deployment, observability, SLO/SLA, runbook)

Проект не имеет права создавать новые типы SPEC локально (§13.3.4).

4.5 Артефакты тестирования

4.5.1 TC — Test Case

TC (`TC-NN`) — артефакт верифицируемого критерия. Покрывает нормативные утверждения BR / SR / SPEC (через `verifies[]` с version pin, §9.4). Жизненный цикл: §4.6.5.

4.5.2 Закрытый список типов TC (`tc-type`)

Закрытый список (§9.5):

`tc-type`	Назначение
`acceptance`	E2E-тесты бизнес-цели для BR (§9.5); runner-family: E2E + AI-валидатор
`ux`	UX-тесты с VLM-judge (§9.6.1) для SPEC-UI
`system`	Системные тесты общего назначения для SR / SPEC-PROC / SPEC-ARCH (§9.5)
`contract`	Контрактные тесты (§9.6.3) для SPEC-API / SPEC-INT / SPEC-DATA
`eval`	Eval-тесты для SPEC-AI с LLM-judge (§9.6.2); judge-модель обязана отличаться от модели реализации

`security`	Security-тесты (§9.6.4) для SPEC-SEC по STRIDE-категориям

4.5.3 Pos / neg парность

Нормативное требование (§9.7): каждое нормативное утверждение покрывается **парой** TC (positive scenario + negative scenario). Единичное TC-покрытие допускается только для утверждений-инвариантов (security STRIDE).

4.6 Статусы жизненного цикла

4.6.1 BR / SR

Закрытый список (§10.5): `draft → approved → verified → accepted → deprecated`. `accepted` — терминальный недеградируемый (§10.4.2, опционально); `deprecated` — терминальный.

4.6.2 TR

Закрытый список (§10.6): `draft → approved → done` ; `obsolete` — альтернативный терминальный.

4.6.3 SPEC

Закрытый список (§10.7): `draft → review → approved → verified` ; `obsolete` — терминальный.

4.6.4 ADAPT

Закрытый список (§10.8): `draft → review → client-ready → answered → approved → frozen`. `frozen` — терминальный immutable; изменения только через delta-ADAPT или errata.

4.6.5 TC

Закрытый список (§10.9): `draft → ready → passing / failing → obsolete`. `passing ⇔ failing` — runner-managed transitions (§10.9.3), не gate-passages.

4.6.6 Backward findings (sub-state в ADAPT)

Закрытый список (§7.4.5): `open → asked-to-client → answered → resolved → frozen` ; `revised` — возврат в `asked-to-client`.

4.6.7 Закрытые списки жизненного цикла — нерасширяемы локально на уровне проекта

Любой статус вне закрытого списка для соответствующего типа артефакта — нарушение соответствия (§10.10.2).

4.7 Quality Gates

Канонический список из §10.3 + §10.4. Проект не имеет права создавать новые gate-типы локально (§10.10.2, §13.3.6).

Gate	Назначение	Статус соответствия
`QG-0` — Approval (§10.3.1)	Approval артефакта для разработки / реализации	**Required**
`QG-1` — Implementation (§10.3.2)	Implementation valid (только для TC `draft → ready`)	**Required**
`QG-2` — Verification (§10.3.3)	Promote артефакта в `verified`	**Required**
`QG-3` — Architecture (§10.4.1)	Approval ADAPT (двойная подпись) / SPEC-ARCH	Опционально (`declared` или `absent`)
`QG-4` — Acceptance (§10.4.2)	Приёмка BR в `accepted`	Опционально

Runner-managed transitions (`ready → passing`, `passing → failing`, и иные) — **не** Quality Gates (§10.9.3).

4.8 Термины носителя

4.8.1 Носитель

Носитель (в полях и коде — `substrate`) — система хранения и версионирования артефактов RENAR. RENAR независим от носителя: нормирует возможности (§4.8.2), а не инструменты.

4.8.2 Возможности V1–V6

Закрытый список из §3.3. Все шесть обязательны абсолютно для соответствия (§13.3.2):

Возможность	Семантика
`V1` — Неизменяемая история	Любое прошлое состояние артефакта восстановимо
`V2` — Атомарная единица изменения	Изменения фиксируются как «всё или ничего»
`V3` — Сравнение различий и рецензирование	Предложенное изменение представимо как diff против опорного состояния и проходит утверждение до интеграции в источник истины
`V4` — Ветвление / набор изменений	Черновики отделимы от утверждённой истины; параллельные изменения независимы

`V5` — Сквозная фиксация версии	Ссылка между носителями фиксирует конкретную версию артефакта
`V6` — Автор и отметка времени	Каждая атомарная единица изменения имеет идентифицируемого автора и отметку времени ≥ секундной точности

4.8.3 Атомарная единица изменения

Изменение носителя (V2), удовлетворяющее свойству «всё или ничего» — нативная для носителя транзакция; промежуточные несогласованные состояния не наблюдаемы наружу. Конкретная реализация (атомарная запись в распределённом VCS, транзакция в document store, иной механизм) специфична для носителя; описание форм выносится в `guide/`.

4.8.4 Фиксация версии

Нативный для носителя механизм (V5), фиксирующий конкретную версию артефакта одного носителя из другого через пару `(artifact-id, version-id)`.

4.8.5 Журнал аудита (audit trail)

Нативная для носителя append-only коллекция событий gate-passage и transitions (§10.13). Каждое событие содержит timestamp, artifact-id, artifact-version, from-status, to-status, gate-id, actor, evidence-refs. Удаление не допускается (V1 retention, §10.13.3).

4.9 Системная иерархия

Закрытый список уровней (§6.7, §6.8):

Уровень	Назначение
`system`	Весь продукт целиком; верхний уровень; редко используется (кросс-подсистемные задачи)
`subsystem`	Подсистема (например, отдельный сервис, фронтенд-приложение)
`module`	Модуль внутри подсистемы

Поле `level` фиксируется во frontmatter артефакта (BR / SR / TR). Иерархия может расширяться вниз через эволюцию подсистема → модуль (§6.9.1) или обратно (§6.9.3).

4.10 Термины происхождения

4.10.1 ai-provenance — каноническая схема

Frontmatter-блок (§11.7.1, RENAR-4 обязательно) фиксирующий происхождение AI-генерируемого артефакта. Настоящая секция — **единственный канонический источник** схемы; chapter-level YAML примеры в §6.5.2, §6.6.2, §8.5, §9.3 ссылаются сюда и **не** определяют независимых полей.

Поле	Обязательно?	Семантика

`ai-provenance.generated-by`	обязательно	Идентификатор модели (`<vendor>-<model>-<version>@<date>`)
`ai-provenance.generated-at`	обязательно	UTC timestamp генерации (ISO-8601)
`ai-provenance.prompt-template`	обязательно	нативный для носителя pointer на prompt template (`<path>@<version>`)
`ai-provenance.context-tokens`	обязательно	Размер контекста на входе (integer)
`ai-provenance.output-tokens`	обязательно	Размер вывода (integer); источник для метрик §12.3
`ai-provenance.human-edits`	обязательно	Boolean — были ли ручные правки после генерации (информационное поле; см. §4.10.1.1)
`ai-provenance.generation-time-ms`	optional	Latency генерации в миллисекундах; рекомендуется на RENAR-5 для cost/latency budget мониторинга (§11.8.1)
`ai-provenance.cost-budget`	optional на RENAR-4, обязательно на RENAR-5	Запланированный budget стоимости генерации
`ai-provenance.cost-actual`	optional на RENAR-4, обязательно на RENAR-5	Фактическая стоимость; источник для §12.3.9 Cost per Approved Requirement

Добавление новых полей в схему — только через формальную процедуру изменения стандарта (§13.9). Локально определённые ai-provenance.* поля проектом-реализацией являются `declared-stricter` extension (§10.10.2) и не нарушают соответствия, но не считаются каноническими.

4.10.1.1 Семантика `human-edits`

`human-edits` — **информационное** поле для traceability и observability, не gating-флаг. Значение `human-edits: true` означает, что артефакт был отредактирован вручную после первичной AI-генерации; оно **не** запускает auto-rejection. Нормативное правило P3 (§9.2) — «инженер не пишет ТС вручную» — нормирует **происхождение**, не последующие правки. Реализация на носителе **может** (`declared-stricter`) дополнительно требовать review для артефактов с `human-edits: true` ; это локальное ужесточение, не часть базового соответствия RENAR-N.

4.10.2 Указание источника (source citation)

Inline pointer в body артефакта (RENAR-4 обязательно, §11.7.1) на источник конкретного нормативного утверждения. Формат специфичен для носителя; типичные паттерны: `[TZ-XXX §Y line Z]` , `[ADAPT-NNN §A.B]` , маркер `derived` с pointer на parent-артефакт.

4.10.3 Цепочка прослеживаемости (traceability chain)

Цепочка артефактов от ТЗ до прогона TC, фиксирующая происхождение каждого утверждения:

```
ТЗ → ADAPT → BR → SR → TR / SPEC → TC.last-run
```

Каждое звено связано через канонические frontmatter-поля (§4.12). Цепочка прослеживаемости — источник истины для ревизии соответствия (§12.5.3).

4.11 Drift классы (закрытый список)

Закрытый список восьми классов нарушений требовательной инфраструктуры. Изменение списка — формальная процедура изменения стандарта (§13.9.3). Нативный для носителя hook (§10.11.1) обязан обнаруживать каждый класс на соответствующей точке контроля.

#	Класс	Что нарушено	Точка контроля
4.11.1	Schema drift	frontmatter артефакта не соответствует обязательной схеме гл. 06/08/09	hook носителя на change-set; RENAR-3+ — blocking (§11.6.1)
4.11.2	Lifecycle drift	Артефакт вне закрытого списка статусов (§4.6) или прошёл forbidden transition (§10.12)	hook носителя на promote-transition
4.11.3	Source-of-truth drift	Реализационный код / производный артефакт расходится с SR / SPEC, на который ссылается через `verifies[].version` (V5)	Reconciliation hook RENAR-4+; регистрация как backward finding в delta-ADAPT
4.11.4	Implementation drift	Реализационный носитель перестал ссылаться на актуальную `version` носителя требований (V5 pin устарел)	Auto-invalidate `verified` (§10.5.4)
4.11.5	Terminological drift	Использование non-canonical термина (§4.14) в нормативном артефакте	hook носителя на change-set; RENAR-4+ — blocking
4.11.6	Order / provenance drift	Артефакт ссылается на источник в более низком статусе чем требует §10.3.1 reference-validation	hook носителя (§10.11.1) блокирует change-set
4.11.7	TC ↔ requirement provenance drift	TC потеряло `verifies[]` ссылку или `last-run.requirement-version` не совпадает с текущей `version`	runner-managed: TC переводится в `failing` (§10.9.3) до повторного прогона
4.11.8	Test-fitting drift	Pass / Fail-критерии TC изменены одновременно с реализационным кодом, чтобы failing TC стал passing без устранения корня (§9.13)	hook носителя через `[test-spec-change]` маркер; единое лицо не может одобрить оба change-set (§10.11.3)

4.12 Термины полей связи (frontmatter, Connection terms)

Канонические имена полей, фиксирующих связи между артефактами:

Поле	Артефакт-источник	Семантика
`parent`	BR / SR	Один родитель в иерархии (BR-NN или SR-NN)
`children[]`	BR / SR	Auto-derived обратное ребро (§6.x)
`implements`	TR	Цепочка реализации (`SR-N` или `BR-N`)
`implements-spec[]`	TR	Реализация спецификаций SPEC-* (§8.6.2)
`constrained-by[]`	SR	Ограничения от SPEC-* (§8.6.1)
`depends-on[]`	SPEC	Граф зависимостей между SPEC (§8.6.3)
`verifies[]`	TC	Закрытый список верифицируемых артефактов с `version` (§9.4)
`verified-by[]`	BR / SR / SPEC	Auto-derived обратное ребро от `verifies[]`
`source.adapt`	BR / SR / SPEC	ADAPT, из которого выведен артефакт
`replaces / replaced-by`	Любой	Замена при deprecation (§10.5.3)
`supersedes`	Новая версия артефакта	Какой артефакт заменяется (взамен «реанимации» obsolete)
`last-run`	TC	Результат последнего прогона runner (§9.12); bot-managed only

Полная схема каждого артефакта — в reference/02-schemas.md.

4.13 Сопоставление с родственными стандартами (Mapping)

4.13.1 Артефакты требований

RENAR canonical	SENAR (RU)	ISO/IEC 29148	BABOK v3	SAFe
`BR` (Business Requirement)	БТ (Бизнес-требование)	Business Requirement	Business Need	Portfolio Epic / Strategic Theme
`SR` (System Requirement)	СТ (Системное требование)	System Requirement / Software Requirement	Solution Requirement (Functional)	Feature
`TR` (Task Requirement)	ТЗ (Требование к задаче)	(нет прямого класса; детализация system /	Solution Requirement (detailed)	Story

		system-element requirement)		
`ADAPT`	(RENAR-extension)	(n/a — formalised bridge artefact)	Stakeholder Requirement workshop output	(n/a)
`TC` (Test Case)	TK	Test Case (verifiable item)	Verification artefact	Story acceptance test
`SPEC-*` (9 types)	(RENAR-extension)	Design Description (subset)	Solution Component (subset)	Enabler tech spec (subset)

4.13.2 Статусы жизненного цикла

RENAR canonical	ISO/IEC 29148	CMMI
`draft`	proposed	identified
`approved`	agreed-to / baselined	committed
`verified`	verified	validated
`accepted`	accepted	accepted
`deprecated` / `obsolete`	retired / superseded	obsolete / superseded

4.13.3 Многоязычный UI

Многоязычные проекты могут отображать каноническую терминологию в UI на язык клиента (RU пример):

English (canonical)	Russian (UI-перевод)
Business Requirement	Бизнес-требование
System Requirement	Системное требование
Test Case	Тест-кейс
Quality Gate	Контрольная точка качества
Acceptance	Приёмка
Verified	Проверено
Approved	Утверждено
Deprecated	Устарело

Это **только UI-перевод**. Frontmatter, ID, имена файлов, нормативные абзацы тела — всегда каноническая латиница / канонический RU из этой главы.

4.14 Запрещённые / устаревшие термины

Закрытый список неканонических терминов; hook носителя RENAR-4+ — blocking при обнаружении в нормативном артефакте (§4.2).

Запрещённый термин	Каноническая замена	Почему
«User Story» как требование	`SR`	Story — единица планирования, не требование; story может реализовывать SR через `implements`
«Use Case» (формально как артефакт)	`SPEC-UI + SR`	Use case mixes UX и behavior; RENAR разделяет SPEC-UI (UX) и SR (поведение)
«Spec» (как родовой термин)	Конкретный `SPEC-*` или `requirement` / `SR`	«Spec» неоднозначно; используем точные термины
«Бизнес-логика»	`SR`	Термин кода, не требований
«Функциональность»	`SR` / `TR`	Слишком широкое
«Фича» / «Feature» (как требование)	`BR` (бизнес-уровень) или `Feature` в SAFe-context (не RENAR canonical)	Mixed Russian / English; RENAR использует BR
«Хотелка»	(никогда)	Договорный документ так не пишется
«Эпик» (как требование)	`BR` (бизнес-уровень)	Epic — единица планирования, не требование

4.14.1 Устаревшие RENAR-specific labels

При migration с pre-v1.0 draft material встречаются устаревшие labels:

Устаревший label	Каноническая v1.0 замена
`UIC` (UI Concept)	`SPEC-UI` (§8.5.6)
`AIC` (AI Concept)	`SPEC-AI` (§8.5.7)
`TS` (Technical Specification)	`SPEC-ARCH` или `SPEC-OPS` в зависимости от содержания
`INT-SR` (Integration SR)	`SR` с `constrained-by: [SPEC-INT-N]`
`INT-TC` (Integration TC)	`TC` с `tc-type: contract`
`TM` (Module/Submodule SR)	`SR` с `level: module` (§6.7)
`QG-0 Context Gate` (старое)	`QG-0 Approval Gate` (канонический v1.0, §10.3.1)
`QG-1 Requirements Gate` (старое)	`QG-1 Implementation Gate` (канонический v1.0, §10.3.2) — semantic shift: ранее approval BR/SR, теперь только TC `draft → ready`
`QG-2 Implementation Gate` (старое)	`QG-1 Implementation Gate` (канонический v1.0, §10.3.2)

`QG-3 Verification Gate` (старое)	`QG-2 Verification Gate` (канонический v1.0, §10.3.3)

Migration существующего носителя требований со старыми labels — отдельный однократный процесс; нативный для носителя hook на change-set должен auto-detect устаревшие labels и предлагать канонические замены.

4.15 Порядок разрешения разногласий (Authority)

При разногласиях по термину порядок обращения:

1. **Эта глава (§4)** — канонический для RENAR-стандарта.
2. **Соответствующая глава стандарта** (06–14) — для специфичной семантики артефактов (например, ADAPT-специфика — в §7).
3. **SENAR §3** (терминология родительского стандарта).
4. **ISO/IEC 29148:2018** — для общеинженерной терминологии requirements.
5. **BABOK v3** — для business-аналитики терминов.
6. **Фиксация через формальную процедуру изменения** стандарта (§13.9.3) — если все вышеперечисленные молчат.

Не использовать как источник терминологии:

- Тикеты ticket-систем (часто противоречивые).
- Чаты команды (slang ≠ канонический).
- Презентации и старые draft-материалы.
- Маркетинговые материалы.

4.16 Связь с другими главами

Глава	Связь
02 Положение в типологии методологий	§2.3 инверсия источника истины + §2.5 независимое от носителя версионирование — фундамент для §4.8 терминов носителя
06 Иерархия требований	frontmatter артефактов BR / SR / TR — §4.3, §4.9, §4.12 связи
07 ADAPT	ADAPT-специфика — §4.3.2, §4.6.4, §4.6.6 backward sub-states
08 Specifications	SPEC-* типы — §4.4, §4.6.3 жизненный цикл SPEC
09 Test cases	TC — §4.5, §4.6.5; pos/neg парность — §4.5.3
10 Жизненный цикл и QG	Quality Gates канонические — §4.7; машины состояний по типам — §4.6; политика закрытого списка — §10.10 параллельно §4.11 / §4.14
03 Версионирование носителя	V1–V6 определения — §4.8.2
11 Maturity model	ai-provenance обязательно на RENAR-4+ — §4.10 (источник критерия — §11.7.1)

12 Metrics	Drift классы §4.11 — источник для метрик типа Reconciliation Findings (§12.3.10)
13 Соответствие	§13.3 обязательные положения ссылаются на каноническую терминологию этой главы
reference/01-glossary.md	Развёрнутые объяснения, anti-patterns, history — ненормативный

05. Роли

> ***Часть RENAR Standard v1.0-draft*** · *← Оглавление*

5.1 Кто за что отвечает

RENAR не заводит свою иерархию ролей — он наследует пять базовых ролей из SENAR §4 и добавляет к ним специализации для работы с требованиями: кто владеет каждым артефактом (ADAPT, BR, SR, SPEC, TR, TC) и кто отвечает за каждый Quality Gate. Здесь же — правило, ради которого глава во многом существует: **двойная подпись ADAPT** (клиент + архитектор), без которой ADAPT не переходит в `approved`.

Глава **не** нормирует нативные для носителя механизмы реализации ролевых проверок (нативный для носителя ACL, V6-идентификаторы автора, нативные для носителя события подписания) — это область главы 3 (возможности носителя) и `guide/03-tool-guide-*.md` (специфичный для носителя инструментарий).

5.2 Базовые роли (SENAR §4)

5.2.1 Закрытый список

RENAR ссылается на пять базовых ролей SENAR §4 как на источник истины. Список закрыт на стороне SENAR; RENAR не добавляет и не удаляет роли.

Роль	Краткая семантика (для RENAR-контекста)
Супервизор	Лицо, инициирующее и контролирующее работу AI-агента; в RENAR — типичный заказчик задач выявления требований, генерации черновиков, рецензирования ADAPT.
AI-агент	Программный участник, выполняющий генерацию и сопровождение артефактов требований (forward интерпретация ADAPT, BR/SR/TR/SPEC/TC, обновление графовых связей при изменениях). В RENAR — **штатный primary author** артефактов (§0.2.1); включает primary-генератора и adversarial-критика. AI-агент **не** является owner (§5.3) и **не** ставит подписей (§5.5).
Архитектор / Tech Lead	Лицо, нормативно ответственное за технические решения и одобрение артефактов требований (QG-0 §10.3.1); подписывающая сторона двойной подписи ADAPT (§5.5). Источник истины — роль SENAR §4 «Architect / Tech Lead»; в русскоязычном корпусе именуется **«Архитектор»**.
Рецензент	Лицо или AI-агент, выполняющий состязательный обзор артефактов; в RENAR — обязательная роль для QG-0 (§10.3.1).
Заинтересованная сторона (Stakeholder)	Внешнее заинтересованное лицо — клиент, представитель клиента с полномочиями, владелец продукта, end-user representative; источник T3 и сторона подписи на стороне клиента (§7.5) и QG-4 (§10.4.2).

Полное определение семантики каждой роли — обязанности, ограничения, взаимодействия — изложено в SENAR §4 и применяется к RENAR без изменений.

> ***Именование роли Архитектора.*** *Везде далее в стандарте, руководстве и приложениях роль SENAR «Architect / Tech Lead» обозначается русским каноническим именем* ***«Архитектор»*** *(например, «Архитектор подтвердил» в §8.3, двойная подпись в §5.5). Это синоним, а не отдельная роль: манифест соответствия обязан использовать нормативное имя SENAR (§5.2.2).*

> ***Именование роли заинтересованной стороны.*** *Везде далее в стандарте, руководстве и приложениях роль SENAR «Stakeholder» обозначается русским каноническим именем* ***«Заинтересованная сторона»*** *(например, «заинтересованная сторона с полномочиями» в §5.3.6, сторона подписи на стороне клиента в §5.5). Это синоним, а не отдельная роль: манифест соответствия обязан использовать нормативное имя SENAR (§5.2.2).*

5.2.2 Запрет переопределения

Реализация **не** имеет права:

- Переименовывать базовые роли SENAR в нормативной части манифеста соответствия (§13.4).
- Объединять две базовые роли в одну так, чтобы исчезала возможность независимого подписания (например, объединение Архитектора и Заинтересованной стороны делает невозможной двойную подпись §5.5 и **не допускается**).
- Добавлять новые базовые роли вне SENAR §4 без формальной процедуры изменения стандарта SENAR.

Локальные для проекта **псевдонимы** (например, локальное название «Lead Engineer» как синоним SENAR Architect / Tech Lead) допустимы в коммуникации, но манифест соответствия (§13.4) обязан использовать нормативные имена ролей SENAR §4.

5.3 Специализации RENAR: ответственность за артефакты

Каждый тип артефакта имеет нормативно зафиксированного owner-а — роль, ответственную за инициирование артефакта, содержательную целостность и one-click promote через QG-0 (§10.3.1).

§	Артефакт	Owner	Тип отв.	Участники QG
5.3.1	**ADAPT**	Архитектор (исполнитель) + Client representative	Совместная — обе роли обязаны для утверждения	QG-0: Архитектор; QG-3 (опц.): двойная подпись (§5.5)
5.3.2	**BR / SR**	Архитектор	Одиночная (Accountable); AI-агент = Responsible primary-генератор	QG-0: Архитектор или authorized role-holder (§5.4); QG-2: автоматический runner; QG-4 (опц.): Заинтересованная сторона
5.3.3	**SPEC-***	Архитектор + технический лидер домена	Совместная по типу SPEC: Архитектор — общая; ТЛ домена — экспертиза	QG-0: Архитектор или authorized role-holder

5.3.4	**TR**	Архитектор (утверждение) + Инженер (execution)	Разделённая	QG-0: Архитектор; QG-2: Инженер + runner
5.3.5	**TC**	Инженер + QA	Совместная — авторство Инженера, QA pos/neg парность (§9.7)	QG-1: Инженер/QA + runner; QG-2: runner с V5 pin
5.3.6	**QG-4 accepted**	Заинтересованная сторона (Клиент + ВП)	Одиночная — Архитектор недостаточен	Только если QG-4 в манифесте (§10.4.2)

5.3.1 Owner ADAPT

ADAPT — единственный артефакт RENAR с **обязательной** совместной ответственностью: одна сторона не может довести ADAPT до `approved` без участия другой (§7.5) — нормативная защита от односторонней интерпретации ТЗ.

5.3.2 Owner BR / SR

Архитектор нормативно ответственен за декомпозицию ADAPT → BR → SR и за согласованность дерева требований. AI-агент — штатный primary-генератор draft BR/SR (§0.2.1, §5.2.1), но **не** owner: owner всегда человек или уполномоченное лицо (§5.4). В RACI: AI-агент — **Responsible** (исполняет), Архитектор — **Accountable** (отвечает за результат). Попытка манифеста объявить AI-агента как Accountable — несоответствующий (§13.3).

5.3.3 Owner SPEC-*

Технический лидер домена — нормативный термин для экспертизы по конкретному типу SPEC (§8.3): ARCH (системный архитектор), API (API designer), DATA (data architect), INT (integration lead), PROC (process owner), UI (UX lead), AI (ML lead), SEC (security lead), OPS (operations lead). В малых проектах одно лицо может совмещать роли ТЛ домена; объединение Архитектора + всех ТЛ домена в одно лицо допустимо при декларации в манифесте (§13.4).

5.3.4 Owner TR

TR имеет разделённую ответственность: одобрение задачи — у Архитектора/уполномоченного лица, выполнение — у Инженера. Инженер не может одобрить собственный TR (нарушение §10.2.2).

5.3.5 Owner TC

AI-агент допустим как primary-генератор TC, но **обязан** проходить QA-проверку перед `ready` (§10.3.2). В проектах без явной QA-роли QA-участником становится инженер, отличный от автора TC.

5.3.6 Owner accepted gate (QG-4)

QG-4 — единственный gate, для которого Архитектор **не является** достаточным участником. Подтверждение post-release бизнес-результата требует заинтересованной стороны с полномочиями

(Клиент + ВП). При отсутствии QG-4 в манифесте gate не применяется, и роль Клиент/ВП на этом этапе жизненного цикла не нормируется.

5.3.7 Политика закрытого списка для owner-распределений

Список §5.3.1-§5.3.6 — закрытый на v1. Локальные для проекта расширения (новый owner для нового типа; перераспределение owner-полномочий между ролями SENAR §4) **не допускаются** без формальной процедуры изменения стандарта (§13.9).

Негативный сценарий: попытка объявить, что owner ADAPT — только Архитектор (без Client representative) — нарушение §5.3.1 и §13.3; манифест несоответствующий.

5.4 Уполномоченное лицо

5.4.1 Нормативное определение

Уполномоченное лицо (authorized role-holder) — лицо с явно делегированными архитекторскими полномочиями для конкретного охвата (один артефакт, один тип артефактов или весь проект). Делегирование фиксируется нативно для носителя (V6 author identifier + ACL §3.3.6); конкретный механизм специфичен для носителя. Уполномоченное лицо — нормативный термин, идентичный по применению в §10.2.2, §10.3.1, §13.5.

5.4.2 Допустимые сценарии

Участник для **QG-0 Approval Gate** (§10.2.2) — BR/SR/SPEC/TR/ADAPT (со стороны архитектора)/TC; **self-assessment** (§13.5) — assessor с ролью `authorized-role-holder` в манифесте.

5.4.3 Недопустимые сценарии

Уполномоченное лицо **не** заменяет `client-signature` в ADAPT (двойная подпись требует заинтересованной стороны на стороне клиента, не уполномоченного лица со стороны исполнителя); **не** заменяет заинтересованную сторону в QG-4 (§10.4.2); **не** заменяет внешнего оценщика в независимой оценке (третьей стороной) (§13.6).

5.4.4 Авторизация на стороне носителя

Делегирование обязано фиксироваться нативно через V6 (§3.3.6): носитель с ACL — через role-based access list; носитель с capability-токенами — через выпуск делегирующего токена; носитель без явной авторизации — через декларацию делегирования в нативном артефакте проекта (отдельный файл с подписью архитектора). Специфичные для носителя детали — `guide/03-tool-guide-*.md`.

5.5 Двойная подпись ADAPT

5.5.1 Нормативное правило

ADAPT переходит в `approved` (QG-3 §10.4.1, §10.8.2) **только** при наличии обеих подписей:

1. `client-signature` — со стороны Client representative (заинтересованная сторона с полномочиями подписания от клиента).
2. `architect-signature` — со стороны Архитектора исполнителя.

Структура полей подписей фиксирована в §7.5; каждая подпись содержит `signed-by`, `role`, `signed-at`, `signature-ref` (нативный для носителя указатель на событие подписания).

5.5.2 Семантика каждой подписи

Подпись	Подтверждает
`client-signature`	Forward интерпретация совпадает с намерением клиента; все backward findings отвечены и ответы — финальные; ADAPT представляет согласованное с клиентом понимание T3.
`architect-signature`	Forward интерпретация технически реализуема; все backward findings в состоянии `resolved` (§7.4.5); декомпозиция в BR / SR / SPEC безопасна для запуска.

Подписи **не** взаимозаменяемы. Архитектор не подтверждает клиентскую семантику; Клиент не подтверждает техническую реализуемость. Оба подтверждения нормативно обязательны.

5.5.3 Негативные сценарии

- **Одна подпись отсутствует**: ADAPT с заполненной только `client-signature` или только `architect-signature` нормативно **не** переходит в `approved`. QG-3 запрещён (§10.8.2); попытка принудительного перехода — нарушение §13.3.
- **Одно лицо в обеих подписях**: реализация, в которой `client-signature.signed-by == architect-signature.signed-by`, нарушает §5.5.1 (две роли требуют двух независимых лиц). Сценарий внутреннего продукта без независимого представителя клиента — вне основной области применения (§1.5.4); манифест такого проекта не заявляет соответствие RENAR-N.
- **Уполномоченное лицо вместо Архитектора**: допустимо (§5.4.2); подпись фиксируется с `role: authorized-role-holder` и `signature-ref` указывает на нативное для носителя доказательство делегирования.
- **Уполномоченное лицо вместо Клиента**: **не допустимо** (§5.4.3); `client-signature` обязана быть от заинтересованной стороны на стороне клиента.

5.5.4 Связь с другими gates

Двойная подпись ADAPT — единственный нормативный случай двойной подписи в RENAR. Остальные gates (§10.3) выполняются одним участником (архитектор, runner, заинтересованная сторона). Это сознательное архитектурное решение: ADAPT — точка согласования с клиентом, остальные gates — внутренние проверки реализации.

5.6 RACI-матрица

Матрица фиксирует Responsible / Accountable / Consulted / Informed по типам артефактов RENAR и каноническим gates. Сокращения: **R** = Responsible (исполняет), **A** = Accountable (одобряет / отвечает за результат), **C** = Consulted (обязан быть проинформирован до решения), **I** = Informed (уведомляется после решения).

5.6.1 Матрица артефактов

Активность	AI-агент	Архитектор	Технический лидер домена	Инженер	QA	Рецензент	
Импорт T3	R	A	—	—	—	C (состязательный)	
Создание ADAPT (forward + backward)	R	A	C	—	—	C (состязательный)	
Утверждение ADAPT (двойная подпись)	—	**A (`architect-signature`)**	—	—	—	C (состязательный)	
Декомпозиция ADAPT → BR	R	A	C	—	—	C	
Декомпозиция BR → SR	R	A	C	—	—	C	
Создание SPEC-*	R	A	**R/A (per type)**	—	—	C	
Создание TR	R	A	C	C	—	—	
Реализация TR (execution)	—	C	C	R	—	—	
Создание TC	R	C	C	R/A	A	—	
Состязательный обзор артефакта	R (AI-критик)	A	—	—	—	R	

5.6.2 Матрица Quality Gates

Соответствие нормативно фиксированной таблице участников §10.2.2.

Gate	Артефакты	Responsible	Accountable	Consulted	Informed
QG-0 Approval Gate	BR, SR, TR, SPEC, TC, ADAPT (со стороны архитектора)	Архитектор или authorized role-holder	Архитектор	AI-критик (Рецензент)	Команда

QG-1 Implementation Gate	TC (`draft → ready`)	Инженер / QA	Инженер	Автоматический runner	Команда
QG-2 Verification Gate	BR, SR, TR, SPEC, TC	Автоматический runner (V5 + V6)	Архитектор	—	Команда
QG-3 Architecture Gate (опционально, ADAPT)	ADAPT (`answered → approved`)	Двойная подпись (Клиент + Архитектор)	Архитектор	AI-критик	Команда
QG-4 Acceptance Gate (опционально, BR)	BR (`verified → accepted`)	Заинтересованная сторона (Клиент + ВП)	ВП	Архитектор	Команда

5.6.3 Закрытый список ролей в RACI

Список ролей-колонок матрицы §5.6.1 / §5.6.2 — закрыт на v1 RENAR:

AI-агент, **Архитектор**, **Технический лидер домена** (per SPEC-type), **Инженер**, **QA**, **Рецензент** (состязательный), **Клиент** (Заинтересованная сторона с полномочиями), **Владелец продукта**.

Локальные для проекта роли (например, «release Manager», «Compliance Officer») допустимы как **Informed** или **Consulted** в локальной practical-RACI реализации, но **не** появляются как **Responsible** или **Accountable** в нормативной матрице соответствия — иначе манифест не соответствующий (§13.3).

5.7 Политика закрытого списка (закрытый список)

5.7.1 Что зафиксировано на v1

Базовые роли SENAR §4 (§5.2.1) закрыты на стороне SENAR; owner-распределение §5.3.1-§5.3.6 закрыто; уполномоченное лицо (§5.4) закрыто; правило двойной подписи ADAPT (§5.5.1) закрыто; ролевые колонки RACI (§5.6.3) закрыты.

5.7.2 Declared-stricter допустимо

Реализация может **ужесточить** требования, явно декларируя в манифесте (§13.4) с маркером `declared-stricter` (§10.10.2): требовать двойную подпись для BR/SR (не только ADAPT); требовать внешний состязательный рецензент на QG-0 для SPEC-SEC и SPEC-AI; запретить совмещение Архитектора и технического лидера домена.

5.7.3 Declared-weaker запрещено

Реализация **не** имеет права объявлять ADAPT утверждённым с одной подписью; инженера-автора TC одобряющим без участника-QA; исполнителя в роли участника QG-4 вместо Заинтересованной

стороны. Манифест с declared-weaker относительно §5 — несоответствующий (§13.8 потеря соответствия).

5.7.4 Путь для расширений

Добавление новой роли (§5.2, §5.3, §5.6.3) или owner-комбинации (§5.3.7) — только через формальную процедуру изменения стандарта (§13.9): исследовательский черновик → публичное обсуждение → повышение minor-версии.

5.8 Перекрёстные ссылки

Источник	Применение
SENAR §4	Закрытый список 5 базовых ролей; semantics и обязанности (нормативный источник)
§7.5 ADAPT approval schema	Структура полей `client-signature` и `architect-signature`
§10.2.2 QG actor table	Нормативное соответствие gate → участник; согласовано с §5.6.2
§10.3.1 QG-0 Approval Gate	Архитектор или authorized role-holder как участник §5.4
§10.4.1 QG-3 Architecture Gate	Требование двойной подписи (§5.5) для ADAPT
§10.4.2 QG-4 Acceptance Gate	Заинтересованная сторона (Клиент + ВП) — §5.3.6
§3.3.6 V6 Author + timestamp	нативная для носителя фиксация авторства для делегирования role-holder §5.4.4
§13.4 Манифест соответствия	Использование нормативных имён ролей §5.2.2
§13.5 Self-assessment	Assessor role enum (architect / authorized-role-holder / external-assessor) — §5.4 anchor
`core/renar-core.md`	Концептуальный обзор стандарта для человека-читателя (ненормативный); roles и signatures полностью нормируются здесь, §5
`guide/03-tool-guide-*.md`	специфичные для носителя механизмы делегирования (§5.4.4) и signature implementations

← Предыдущая: 04. Термины и определения · **Оглавление** · **Следующая: 06. Иерархия требований →**

06. Иерархия требований

> ***Часть RENAR Standard v1.0-draft*** · *← Оглавление*

> ***Плотная глава:*** *перед frontmatter — guide/00 quickstart; плотность глав — reference/09.*

6.1 Три типа требований и три уровня

У требования три высоты, и путать их нельзя. **Бизнес** хочет результата: «клиент сам восстанавливает пароль, чтобы разгрузить поддержку». **Система** должна вести себя так, чтобы этот результат стал возможен: «по запросу с подтверждённого адреса система отправляет ссылку сброса со сроком 30 минут». **Инженер** берёт это в работу одной задачей: «сделать сброс пароля с ограничением три попытки в час». Три разных вопроса — зачем, что и как именно, — и каждому RENAR даёт свой тип артефакта: **BR**, **SR**, **TR**.

Типов ровно три, список закрыт, и связаны они в дерево: один SR раскрывает один BR, одна TR — один SR. Сверху накладываются три уровня масштаба — система, подсистема, модуль; они определяют, какие из трёх типов вообще уместны (у модуля нет своего бизнес-владельца — значит, нет и BR). На этой оси держится вся иерархия источника истины из главы 2 §2.3: ТЗ → ADAPT → BR / SR / SPEC → TR → TC. Структура системы описывается параллельно — спецификациями SPEC-* (глава 8), на которые BR / SR / TR ссылаются через типизированные рёбра графа.

Положения главы нормативны. Закрытые списки типов и уровней — обязательные положения (глава 13); расширить их можно только формальной процедурой изменения стандарта.

Глава опирается на ISO/IEC/IEEE 29148:2018 «Requirements engineering» в части понятий business / system / task requirements и принципов traceability, но фиксирует закрытый список ровно из трёх типов оси требований v1.0 и явно отстраивается от свободно расширяемого набора типов, характерного для классических подходов.

6.2 Закрытый список трёх типов оси требований

6.2.1 Нормативная формулировка

Ось требований RENAR v1.0 содержит ровно три типа: BR, SR, TR. Список закрыт. Новые типы добавляются только через формальную процедуру изменения стандарта (глава 13).

Тип	Расшифровка	Вопрос	Содержит	Не содержит
BR	Business Requirement	Кто, что и зачем?	Бизнес-цель, роль, ценность	Технологии, экраны, контракты, поля данных
SR	System Requirement	Что делает система?	Поведение системы, ограничения	Имена таблиц, фреймворки, конкретные структуры
TR	Task Requirement	Что именно реализовать?	Конкретика реализации: поля, условия, ошибки	Архитектурные решения

6.2.2 Дерево родителей

```
BR
 └── SR              # parent = BR (единственный)
      └── TR         # parent = SR (единственный)
                ↓
          Goal + Acceptance Criteria
          в системе управления задачами
```

`SR.parent` — единственный BR. `TR.parent` — единственный SR. Это **дерево**, не граф; множественные родители на оси требований запрещены. Граф связей — между требованиями и спецификациями (§6.10).

6.2.3 Что не является типом оси требований

Артефакты, исторически встречавшиеся в проектах под именами UIC (UI Concept), AIC (AI Concept), INT-SR (интеграционное требование), TS (техническая спецификация), **не являются типами оси требований в v1.0**. Они перенесены в параллельную ось спецификаций как соответствующие типы SPEC (см. §8.3 закрытый список 9 типов SPEC и §8.7 миграция):

Legacy артефакт	RENAR v1.0 тип
UIC	SPEC-UI
AIC	SPEC-AI
INT-SR	SPEC-INT
TS (architecture / data / API / process / security / ops)	SPEC-ARCH / SPEC-DATA / SPEC-API / SPEC-PROC / SPEC-SEC / SPEC-OPS

SPEC-* связан с осью требований не как «ещё один тип требования», а как параллельная ось с типизированными рёбрами `SR.constrained-by[]` и `TR.implements-spec[]` (§6.10).

Тест-кейсы (TC) — отдельный класс артефактов верификации, нормируемый главой 9. TC не являются требованиями: они проверяют поведение, описанное в BR / SR / SPEC.

6.3 Системы, подсистемы, модули

Уровни организационной декомпозиции — закрытый список трёх элементов: **система**, **подсистема**, **модуль**. Каждому элементу соответствует допустимый набор типов требований (см. §6.4).

6.3.1 Система

Система — верхний уровень иерархии. Целостный продукт или платформа, которая поставляется и эксплуатируется как единое целое и за которую отвечает организация.

Признаки системы:

- единый владелец со стороны бизнеса (Владелец продукта, директор, заказчик);
- поставляется и эксплуатируется как единое целое;

- клиент видит и оценивает систему в целом, а не её части по отдельности;
- единый верхнеуровневый ТЗ-документ и один корневой ADAPT.

Допустимые типы требований: **BR, SR, TR**.

6.3.2 Подсистема

Подсистема — крупный самостоятельный компонент системы, обладающий собственной бизнес-ценностью или отдельным бизнес-владельцем.

Подсистема выделяется, если выполняется **хотя бы одно** из условий:

Условие	Пример
Своя команда или технический владелец	Команда фронтенда vs команда AI-pipeline
Отдельная база данных или независимо разворачиваемый сервис	Изолированный микросервис с отдельным деплоем
Возможность быть заменённой независимо от других	Аналитический модуль заменяется без изменения операционного контура
Другой бизнес-домен или отдельная заинтересованная сторона	Финансовый директор vs операционный директор
Добавляется позже как отдельная инициатива с отдельным бюджетом	Партнёрская программа

Ключевой нормативный критерий: существует ли отдельный человек со стороны бизнеса, отвечающий за ценность этой части системы?

- да → подсистема, BR оправданы;
- нет → модуль, только SR.

Допустимые типы требований: **BR (если есть своя заинтересованная сторона) + SR + TR**.

6.3.3 Модуль

Модуль — техническое деление внутри подсистемы. Реализует часть поведения подсистемы, но не имеет самостоятельной бизнес-ценности.

Признаки модуля:

- отсутствует отдельная бизнес-заинтересованная сторона;
- не существует и не используется в отрыве от своей подсистемы;
- выделяется по техническому принципу: функциональная область, слой, домен;
- не упоминается отдельно в ТЗ верхнего уровня.

Допустимые типы требований: **SR + TR**. BR не создаётся.

6.3.4 Сводная таблица

	Система	Подсистема	Модуль
Бизнес-заинтересованная сторона	да	да (свой)	нет

Независимое развёртывание	возможно	да	нет
Упоминается в ТЗ отдельно	да	да	редко
Существует отдельно	да	возможно	нет
BR	да	да (если своя заинтересованная сторона)	нет
SR	да	да	да
TR	да	да	да

6.3.5 Эволюция «модуль → подсистема»

Граница между модулем и подсистемой не является навечно зафиксированной. Если модуль вырос, получил собственную команду или бизнес-владельца — он становится подсистемой и получает BR. Нормативно: **BR пишется в момент появления бизнес-владельца, не авансом**.

Обратная эволюция (подсистема → модуль) — также допустима, если бизнес-владелец ушёл, бизнес-ценность стала производной; в этом случае BR подсистемы получает статус `deprecated` (§6.5.4), а его SR переподчиняются BR родительской системы.

6.4 Допустимые типы требований по уровням декомпозиции

Уровень	BR	SR	TR	Пояснение
Система	обязательно	обязательно	обязательно	Верхний BR — обязателен (без бизнес-цели нет проекта)
Подсистема	опционально	обязательно	обязательно	BR — только при наличии собственной заинтересованной стороны
Модуль	не допускается	обязательно	обязательно	BR запрещён нормативно

Нормативное обоснование запрета BR на уровне модуля: бизнес-требование без бизнес-заинтересованной стороны и без независимой бизнес-ценности приводит к ложной декомпозиции и порождает «технические BR», не отличимые от SR. Это размывает иерархию источника истины (глава 2 §2.3).

6.5 BR — Business Requirement

6.5.1 Нормативное определение

BR фиксирует **что нужно бизнесу и зачем**. Описывает роль, действие и бизнес-ценность без отсылок к технологиям, экранам, контрактам, структурам данных.

6.5.2 frontmatter BR (обязательные поля)

```yaml
---
id: BR-NN                              # immutable; NN sequential в рамках scope
title: "<short, descriptive>"
type: BR
slug: "<kebab-case>"                   # auto-derived

# === Scope (mandatory) ===
level: system | subsystem              # BR на уровне модуля запрещён (§6.4)
scope:
  system: "<system-id>"
  subsystem: "<subsystem-id>"          # null если level=system

# === Lifecycle (mandatory) ===
status: draft | approved | verified | deprecated
owner: "<role / responsible person>"

# === Source: provenance (conditional, см. §7.4.1) ===
# source.adapt — mandatory если ADAPT существует (gap при конвертации ТЗ → RENAR обнаружен);
# source.tz-section — mandatory всегда. Минимум один источник всегда присутствует.
source:
  adapt: ADAPT-NNN                     # conditional: present если ADAPT создавался для этого ТЗ
  adapt-section: "Forward §N"          # mandatory если adapt present
  tz-section: "§N.N"                   # mandatory всегда — primary provenance
  adversarial-review-ref: "<нативная для носителя ссылка>"   # conditional: present если source.adapt отсутствует — evidence verdict «no findings» (§7.4.1.2)

# === Межуровневая связь BR подсистемы → BR системы (см. §6.8.2) ===
# Recommended на v1.0; mandatory на v1.1+ при level=subsystem И parent system имеет approved BR.
# Не parent-edge — отдельный тип ребра графа связей (см. §6.8.3).
implements:                            # массив; substrate-agnostic
  - id: BR-NN                          # ID BR родительской системы
    scope:
      system: "<system-id>"            # обязательно для cross-system ссылки
    rationale: "<short>"               # опционально; ссылка на ADAPT§ при наличии

# === Граф связей (auto-managed) ===
children: []                           # auto-derived; SR со ссылкой parent.id=<этот BR>
implemented-by: []                     # auto-derived; BR подсистем со ссылкой implements[].id=<этот BR>
verified-by: []                        # auto-derived; TC верифицирующие через SR

# === AI provenance (обязательно на RENAR-4+; canonical schema — §4.10.1) ===
ai-provenance:
  generated-by: "<vendor>-<model>@<date>"
  generated-at: "<ISO-8601>"
  prompt-template: "<template-path>@<version>"
  context-tokens: integer
  output-tokens: integer
```

```
  human-edits: boolean
  # optional на RENAR-4, обязательно на RENAR-5 (см. §4.10.1):
  # cost-budget, cost-actual, generation-time-ms

# === Замена (mandatory если применимо) ===
replaces: "<old-id>"
replaced-by: "<new-id>"
deprecated-date: "<ISO date>"
---
```

Поле `source.tz-section` обязательно всегда. Поле `source.adapt` — условное: оно присутствует, когда конвертация ТЗ → RENAR потребовала ADAPT (§7.4.1.1), и опускается, когда состязательный рецензент вынес вердикт «no findings, no clarifications» (§7.4.1.2). Если `source.adapt` опущено, обязательно поле `source.adversarial-review-ref`: оно хранит свидетельство этого вердикта для аудита. Hooks жизненного цикла (§7.4.1, §10.11.1) проверяют оба случая: (1) если `source.adapt` присутствует — что ADAPT в статусе `approved` или выше; (2) если `source.adapt` опущено — что `source.adversarial-review-ref` присутствует, а свидетельство доступно аудитору по запросу.

Поле `parent` отсутствует в BR: BR — корневой узел дерева требований. Для межуровневой связи между деревом подсистемы и деревом системы используется отдельное поле `implements[]` (см. §6.8.2): это **не** parent-edge, а типизированная межуровневая декларация «BR подсистемы раскрывает указанные BR системы». Запрет множественных parents (§6.8.3) на `implements[]` не распространяется.

6.5.3 Body BR (обязательные разделы)

Раздел	Обязательность	Содержание
Потребность	обязательно	Кто (роль), что (действие), зачем (бизнес-цель). Формулировка в одном предложении.
Критерии успеха	обязательно	Измеримые результаты (3–7 пунктов); каждый поддаётся независимой проверке.
Контекст	обязательно	Откуда взялось требование (со ссылкой на раздел ADAPT), какие альтернативы рассматривались.
Ограничения	опционально	Бизнес-ограничения (бюджет, сроки, регуляторика), не технические.

Технические детали (UI, API, схема данных) в BR запрещены — для этого существуют типы SPEC (глава 8) и SR.

6.5.4 Статусы BR

Статус	Смысл	Триггер перехода
`draft`	В проработке	Создаётся автором

`approved`	Утверждено, можно декомпозировать в SR	После QG-0 (глава 10)
`verified`	Все производные SR / TR / TC выполнены, бизнес-результат подтверждён	После QG-2; все `verified-by` TC имеют `last-run.result = pass` на текущей версии
`deprecated`	Устарело; заменено другим BR или утратило актуальность	Архитектором / Владельцем продукта, обязательно с `replaced-by` (если замена)

BR в статусе `deprecated` **не удаляется** — остаётся как исторический след для аудита.

6.6 SR — System Requirement

6.6.1 Нормативное определение

SR фиксирует **что делает система** (на уровне системы, подсистемы или модуля). Описывает наблюдаемое поведение и ограничения. Не описывает имена таблиц, фреймворков и конкретных структур данных — это ответственность SPEC (глава 8).

6.6.2 frontmatter SR (обязательные поля)

```yaml
---
id: SR-NN                              # immutable
title: "<short, descriptive>"
type: SR
slug: "<kebab-case>"

# === Scope (mandatory) ===
level: system | subsystem | module
scope:
  system: "<system-id>"
  subsystem: "<subsystem-id>"          # null если level=system
  module: "<module-id>"                # null если level ≠ module

# === Lifecycle (mandatory) ===
status: draft | approved | verified | deprecated
owner: "<role / responsible person>"

# === Parent (mandatory) ===
parent:
  id: BR-NN                            # единственный родитель

# === Source: provenance (conditional, см. §7.4.1) ===
# Те же правила что для BR: source.adapt conditional; source.tz-section mandatory
всегда;
# source.adversarial-review-ref mandatory если source.adapt omitted.
source:
  adapt: ADAPT-NNN                     # conditional
  adapt-section: "Forward §N"          # mandatory если adapt present
```

```yaml
  tz-section: "§N.N"                  # mandatory всегда
  adversarial-review-ref: "<нативная для носителя ссылка>"   # mandatory если
adapt omitted

# === Граф связей (mandatory ones + auto-managed) ===
constrained-by:                       # типизированные рёбра к SPEC (глава 8)
  - SPEC-UI-NN
  - SPEC-API-NN
  - SPEC-DATA-NN
children: []                          # auto-derived; TR со ссылкой parent.id=
<этот SR>
verified-by: []                       # auto-derived; TC верифицирующие SR

# === AI provenance (обязательно на RENAR-4+) ===
ai-provenance:
  generated-by: "<vendor>-<model>@<date>"
  prompt-template: "<template-path>@<version>"
  context-tokens: integer
  output-tokens: integer
  human-edits: boolean

# === Замена (mandatory если применимо) ===
replaces: "<old-id>"
replaced-by: "<new-id>"
deprecated-date: "<ISO date>"
---
```

Ключевые поля. `parent.id` — единственный BR; это дерево родителей. `constrained-by[]` — типизированные ссылки на SPEC-*; это **граф**, не дерево. SR может ссылаться на произвольное количество SPEC любых типов; SPEC, в свою очередь, может быть `referenced-by` множеством SR (глава 8 §8.2).

6.6.3 Body SR (обязательные разделы)

Раздел	Обязательность	Содержание
Требование	обязательно	Одно предложение нормативной формы: «Система должна …» (модальность — по конвенции §0.5: «должна» / «обязана» = обязательно).
Поведение	обязательно	Детальное описание наблюдаемого поведения; функциональные сценарии.
Ограничения	обязательно если применимо	Нефункциональные ограничения (производительность, безопасность); полные ограничения — в `constrained-by[]` SPEC.
Связь с SPEC	обязательно если присутствует `constrained-by[]`	Краткое объяснение, какие аспекты поведения нормированы какими SPEC.

6.6.4 Статусы SR

Идентичны статусам BR (§6.5.4) — `draft → approved → verified → deprecated`. Переход `approved → verified` — после QG-2 (глава 10); требует, чтобы все `verified-by` TC имели `last-run.result = pass` на текущей версии SR.

6.7 TR — Task Requirement

6.7.1 Нормативное определение

TR — атомарная единица работы исполнителя. Фиксирует **что именно реализовать** в рамках одного SR. TR декомпозирует SR до уровня, пригодного для прямой реализации (одна задача — одна TR).

6.7.2 frontmatter TR (обязательные поля)

```yaml
---
id: TR-NN                              # immutable
title: "<short, descriptive>"
type: TR
slug: "<kebab-case>"

# === Scope (mandatory) ===
level: system | subsystem | module   # system — редко, кросс-подсистемные задачи
scope:
  system: "<system-id>"
  subsystem: "<subsystem-id>"        # null если level=system
  module: "<module-id>"              # null если level ≠ module

# === Lifecycle (mandatory) ===
status: draft | approved | done | obsolete
owner: "<assignee role / agent>"

# === Parent (mandatory) ===
parent:
  id: SR-NN                          # единственный родитель

# === Source: цепочка прослеживаемости (auto-derived from parent SR) ===
# TR не имеет собственного source — наследует от parent SR (§6.7.5).
# Если parent SR.source.adapt omitted (§7.4.1) — этот факт также наследуется.
source:
  adapt: ADAPT-NNN                   # auto-derived from parent SR; может быть
omitted
  sr-version: "<version-ref>"        # pinning к версии SR (возможность носителя
V5; глава 3)

# === Граф связей ===
implements-spec:                     # типизированные рёбра к SPEC
  - SPEC-API-NN
```

```
  - SPEC-UI-NN
verified-by: []                    # auto-derived; TC верифицирующие через SR

# === Goal + Acceptance Criteria ===
goal: "<one-sentence outcome>"
acceptance-criteria:
  - "<numbered, falsifiable, без двусмысленности>"
  - "..."

# === AI provenance (обязательно на RENAR-4+) ===
ai-provenance:
  generated-by: "<vendor>-<model>@<date>"
  human-edits: boolean
---
```

Ключевые поля. `parent.id` — единственный SR. `implements-spec[]` — типизированные рёбра к SPEC; конкретизирует, какие SPEC должны быть учтены при реализации именно этого TR (подмножество `constrained-by[]` родительского SR или его расширение типами SPEC, не указанными у SR прямо). `acceptance-criteria` — закрытый нумерованный список опровержимых утверждений.

6.7.3 Body TR (обязательные разделы)

Раздел	Обязательность	Содержание
Goal	обязательно	Один параграф; результат, который TR делает наблюдаемым.
Acceptance Criteria	обязательно	Нумерованный список; каждый пункт опровержим; покрывает положительные и отрицательные сценарии.
Scope	обязательно	Что входит / что не входит в TR (соответствует SENAR Rule 2).
Ссылки	обязательно если применимо	На SPEC из `implements-spec[]` и разделы родительского SR.

6.7.4 Статусы TR

Статус	Смысл	Триггер
`draft`	TR создан, AC ещё не финализированы	Авторство
`approved`	AC утверждены, разрешён старт работы	QG-0 (глава 10): goal + AC присутствуют
`done`	AC верифицированы, TC прошли	QG-2; `verified-by` TC `pass`
`obsolete`	TR утратил актуальность до завершения (например, SR изменился)	Архитектором, обязательно с пометкой

6.7.5 TR не ссылается на ADAPT напрямую

Исполнитель TR работает в рамках SR / SPEC и **не обращается к ADAPT напрямую** — все нужные интерпретации ТЗ уже зафиксированы в approved ADAPT и протянуты в SR / SPEC через `source.adapt` (глава 7 §7.7.3). Если исполнитель обнаружил неясность в SR — это сигнал либо для новой обратной находки в ADAPT (если корень неясности в ТЗ), либо для уточнения SR (если корень в декомпозиции).

6.8 Расширенная иерархия для составных систем

Базовая схема `BR → SR → TR` справедлива для большинства проектов. Для составных систем стандарт нормирует два варианта расширенной иерархии.

6.8.1 Подсистема — техническое деление, не самостоятельный продукт

BR относятся к системе в целом; подсистемы — техническое деление по командам, компонентам или другим архитектурным границам:

```
BR (система)
 └── SR (система)          # опционально, если есть кросс-подсистемные SR
      └── SR (подсистема)
           └── TR

SPEC-INT (между подсистемами)    # параллельная ось; см. главу 8
```

`SPEC-INT` не принадлежит ни одной из подсистем — это спецификация интеграции на уровне системы.

6.8.2 Подсистема — самостоятельный продукт со своей заинтересованной стороной

Подсистема имеет собственный BR со своим бизнес-владельцем:

```
BR (система)
 └┄┄ BR (подсистема)       # ┄┄ implements-edge (см. ниже), НЕ parent-edge
      └── SR (подсистема)
           └── TR
```

`└┄┄` между BR (система) и BR (подсистема) обозначает **типизированное межуровневое ребро** **`implements`** (§6.5.2): BR подсистемы декларирует, какие BR родительской системы он раскрывает и реализует. Это **не parent-edge** дерева требований: `BR (подсистема).parent` остаётся отсутствующим, и каждая такая подсистема является корневым узлом своего дерева.
`implements` — отдельный тип ребра графа связей, симметричный паре `constrained-by[] ↔ referenced-by[]` для SPEC (§6.10.2).

Нормативное правило `implements[]`:

Уровень	Сценарий	Правило
Recommended на v1.0 / обязательно на v1.1+	`BR.level = subsystem` И `scope.system` имеет ≥1 approved BR	`implements[]` обязан содержать ≥1 ссылку на applicable BR родительской системы
Permitted	`BR.level = subsystem` И родительская система — контейнер без собственных BR (охват организационного уровня)	`implements[]` опускается; обоснование фиксируется в разделе `Контекст` со ссылкой на ADAPT§
Запрещено	`BR.level = system`	`implements[]` не применяется (system BR — корень всей иерархии охвата)

Hooks жизненного цикла (глава 10 §10.11) обязаны:

- Проверять существование target BR (по `id + scope.system`) при approve BR подсистемы.
- Проверять, что target BR в статусе `approved` или выше; ошибочный draft target — фатально.
- Выявлять циклы цепочек `implements`; цикл — фатально.
- При deprecate target BR (§6.5.4) — генерировать cascade-warning для всех `implemented-by[]` (не cascade-deprecate; решение об эволюции зависимых BR — у Архитектора).

Машиночитаемая цепочка прослеживаемости в §6.8.2 восстанавливается через `implements`-ребро (§6.10.3) — асимметрия с §6.8.1 устраняется.

Связь подсистемы с общим ADAPT системы сохраняется через `source.adapt` (если применимо); `implements[]` и `source.adapt` — независимые поля и могут указывать на разные узлы графа.

6.8.3 Запрет множественных родителей

Стандарт не допускает множественное `parent` для SR или TR. Кросс-функциональные требования, которые могли бы выглядеть как «дочерние сразу двух SR», нормируются одним из двух способов:

- разделяются на несколько SR, каждое с единственным parent BR;
- декомпозируются в SR более высокого уровня (родительская подсистема или система), от которой эти кросс-функциональные сценарии и зависят.

Поле `BR.implements[]` (§6.5.2, §6.8.2) — **не parent-edge** и под действие §6.8.3 не подпадает: один BR подсистемы может раскрывать несколько BR системы (cardinality 0..N). Это сознательная разница типизации рёбер графа связей: parent — единственный, межуровневые декларации — множественные.

6.9 Эволюция иерархии

6.9.1 Модуль → подсистема

Сценарий из §6.3.5: модуль получает бизнес-владельца. Нормативная последовательность:

1. Появление бизнес-владельца фиксируется через обратную находку в ADAPT (категория `scope` — изменение границ работ; глава 7 §7.4.4).
2. После approved delta-ADAPT — модуль переводится в статус подсистемы; создаётся BR подсистемы с `source.adapt: <delta-ADAPT>`.
3. Существующие SR модуля сохраняются (неизменяемые ID); поле `parent` SR обновляется на новый BR подсистемы атомарным изменением.
4. TR / TC, ссылающиеся на эти SR, не требуют изменений (parent SR неизменен).

6.9.2 Запрет авансовой иерархии

Создание BR подсистемы «на вырост», без существующего бизнес-владельца — нарушение стандарта. BR без заинтересованной стороны превращается в «технический BR», размывающий инверсию источника истины и подменяющий собой SR. Hooks жизненного цикла (глава 10) обязаны блокировать переход BR в `approved`, если в ADAPT не зафиксирован выявленный бизнес-владелец.

6.9.3 Подсистема → модуль

Симметричный сценарий §6.3.5: подсистема утратила бизнес-владельца или бизнес-ценность стала производной от родительской системы. Нормативная последовательность:

1. Утрата бизнес-владельца / переоценка бизнес-ценности фиксируется через обратную находку в ADAPT (категория `scope`; глава 7 §7.4.4).
2. После approved delta-ADAPT — BR подсистемы переводится в статус `deprecated` с указанием причины в `Контексте` (бизнес-владелец отозван / бизнес-ценность поглощена системой). BR не удаляется (immutable IDs, V1).
3. SR подсистемы сохраняются (неизменяемые ID); поле `parent` SR обновляется атомарным изменением на BR родительской системы (или на BR другой подсистемы, если область SR относится к ней).
4. Подсистема переименовывается в модуль на уровне области и схемы хранения (§6.11.2); существующие IDs SR / TR остаются неизменными.
5. TR / TC, ссылающиеся на эти SR, не требуют изменений.

Если ни одна BR родительской системы не покрывает поведение SR — это сигнал того, что подсистема в действительности не утратила самостоятельную бизнес-ценность; обратная эволюция в этом случае невозможна, и BR подсистемы остаётся `approved`.

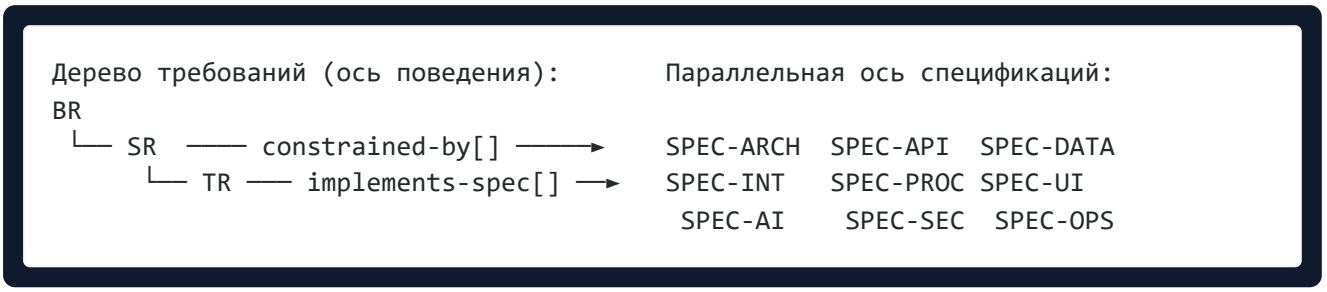
6.10 Связь иерархии с ADAPT и SPEC

Стандарт фиксирует связи между осью требований, ADAPT и параллельной осью SPEC через нормативные поля frontmatter и типизированные рёбра графа связей.

6.10.1 Связь с ADAPT

`BR.source.adapt`, `SR.source.adapt`, `SPEC-*.source.adapt` — условные ссылки на ADAPT, из которого артефакт был выведен (глава 7 §7.7.1): присутствуют, когда ADAPT создавался; при вердикте «no findings» ADAPT не существует, и вместо ссылки обязательно поле `source.adversarial-review-ref` (§7.4.1). TR не имеет прямого `source.adapt` — наследует его через `parent SR` (§6.7.5).

6.10.2 Граф связей с SPEC



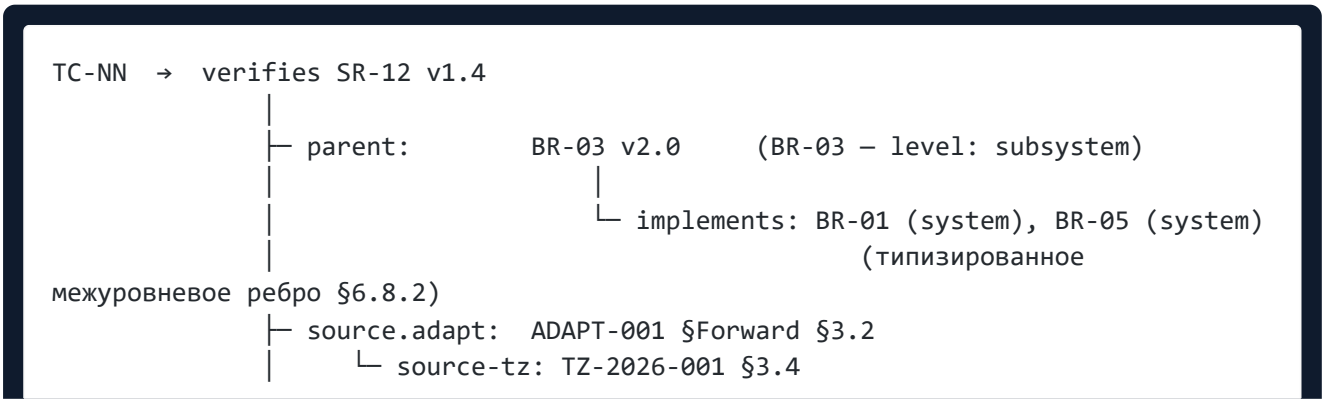
Ребро	Тип	Обязательность
`SR.constrained-by[] → SPEC-*`	Граф (множественное)	Optional; присутствует при наличии нормирующих SPEC
`TR.implements-spec[] → SPEC-*`	Граф (множественное)	Optional; конкретизирует SPEC для исполнителя
`SPEC-*.referenced-by[] → SR / TR`	Auto-derived inverse	Авто-вычисляется нативной для носителя индексацией
`SPEC-*.depends-on[] → SPEC-*`	Граф между SPEC	См. глава 8 §8.2

Связь оси требований и оси SPEC — нормативная: ни один SR с нетривиальным поведением UI / API / данных не должен оставаться без `constrained-by[]` (отсутствие — сигнал либо для написания SPEC, либо для обоснования отсутствия в Контексте SR).

6.10.3 Полная цепочка прослеживаемости (read-side)

ADAPT — реактивный артефакт (§7.4.1): создаётся только когда конвертация ТЗ → RENAR порождает разрыв между языками. Цепочка прослеживаемости соответственно имеет **два валидных варианта**, выбираемых в зависимости от того, существует ли ADAPT для конкретного ТЗ.

Вариант А — когда ADAPT создавался (`source.adapt` present):



```
              ├─ constrained-by: SPEC-UI-04, SPEC-API-02
              └─ children:      TR-101, TR-102, TR-103
                                    └─ implements-spec: SPEC-API-02
                                    └─ verified-by: TC-NN
```

Вариант B — когда ADAPT не создавался (`source.adapt` omitted; состязательный рецензент вынес «no findings» вердикт, §7.4.1.2):

```
TC-NN  →  verifies SR-12 v1.4
              │
              ├─ parent:        BR-03 v2.0     (BR-03 — level: subsystem)
              │                     │
              │                     ├─ implements: BR-01 (system), BR-05 (system)
              │                     └─ source.tz-section: TZ-2026-001 §3.4
              │                        source.adversarial-review-ref: <verdict
evidence ref>
              ├─ source.tz-section: TZ-2026-001 §3.4   (нет ADAPT для этого ТЗ)
              │  source.adversarial-review-ref: <verdict evidence ref>
              ├─ constrained-by: SPEC-UI-04, SPEC-API-02
              └─ children:      TR-101, TR-102, TR-103
                                    └─ implements-spec: SPEC-API-02
                                    └─ verified-by: TC-NN
```

Оба варианта **машиночитаемы**. В варианте B путь восстанавливается через `source.tz-section` напрямую; свидетельство `adversarial-review-ref` фиксирует, кем и когда было задекларировано «no findings» (V6 author + timestamp), и доступно по запросу аудитора (§13.5).

Для сценария «подсистема — самостоятельный продукт» (§6.8.2) цепочка в обоих вариантах содержит ребро `implements` между BR подсистемы и BR родительской системы — это восстанавливает машиночитаемую прослеживаемость, симметричную сценарию §6.8.1.

Дезавуированный ADAPT в цепочка прослеживаемости. Когда ADAPT переходит в `superseded` (§7.6.4, §10.8.5), все производные BR / SR / SPEC с `source.adapt` на него обязаны быть либо перенаправлены на дезавуирующий ADAPT, либо пере-выведены. Висячая ссылка `source.adapt` на ADAPT в статусе `superseded` делает цепочку прослеживаемости **невалидной** (read-side): аудит не должен приводить к дезавуированному источнику интерпретации как к действующему. Сам `superseded` ADAPT сохраняется для аудита (V1) и достижим по ребру `superseded-by` от дезавуирующего ADAPT — но не как `source.adapt` действующего требования. Обеспечение соблюдения — `adapt-supersession` validation (§10.11.1).

6.11 Схема хранения

Требования хранятся в подпапках носителя требований. Нативная для носителя реализация хранения специфична для носителя (см. guide/03 для distributed VCS; guide/04 для document-oriented store).

6.11.1 На уровне системы

```
[requirements-substrate]/        # корень носителя требований (layout — guide/03
или guide/04)
  br/                            # BR-NN-*.md
  sr/                            # SR-NN-*.md, level=system
  tr/                            # TR-NN-*.md, level=system (редко)
  specs/                         # SPEC-* (глава 8)
  adapt/                         # ADAPT (глава 7)
  tz/                            # immutable исходники T3
  REQUIREMENTS.md                # auto-generated index
```

6.11.2 На уровне подсистемы / модуля

```
[subsystem-substrate]/           # scope подсистемы внутри носителя требований
  br/                            # если у подсистемы своя заинтересованная
сторона
  sr/
  tr/
  modules/
    [module-substrate]/
      sr/                        # модуль имеет только SR + TR
      tr/
  specs/                         # глава 8
  adapt/
  REQUIREMENTS.md
```

6.11.3 REQUIREMENTS.md — auto-generated index

`REQUIREMENTS.md` — auto-generated реестр всех BR / SR / TR в области: ID, тип, уровень, заголовок, статус, parent, ссылка на файл. Помечается нативным для носителя флагом auto-generated. Триггеры перегенерации — каждое изменение frontmatter или каждый approve / verify gate (глава 10).

6.12 Контрольные точки качества для оси требований

Детальные определения gates — в главе 10. Краткая сводка для BR / SR / TR:

Gate	Применим к	Предусловие	Постусловие
QG-0 (Approval)	BR / SR / TR (`draft → approved`)	frontmatter валиден, обязательные поля заполнены, идентификатор уникален (V1); `source.adapt` , если присутствует, указывает на approved ADAPT, иначе присутствует `source.adversarial-review-ref` (BR / SR); `parent` указывает на approved BR / SR (SR); body разделы соответствуют §6.5.3 / §6.6.3; состязательный обзор произведён	Артефакт переведён в `approved` ; immutable до версии следующего change; разрешена декомпозиция в дочерние артефакты

QG-2 (Verification)	BR / SR / TR (`approved → verified` / `done`)	Все `verified-by` TC `pass` на текущей версии артефакта	Артефакт `verified` (BR/SR) или `done` (TR); цепочка до родительского BR — обновляется к verified, если все дети verified

QG-1 (Implementation) к оси требований **не применяется**: это отдельный gate только для TC (§10.3.2) — промежуточного «QG-1 implementation» для BR / SR / TR не существует, переход `approved → verified` / `done` управляется единым QG-2. TR переходит `draft → approved` тем же QG-0 единым шагом (frontmatter + goal + AC). `ready` и аналогичные термины — не статусы оси требований и не присутствуют в машине состояний `draft → approved → verified | done | obsolete | deprecated` (см. §6.5.4 / §6.6.4 / §6.7.4).

Hooks носителя (глава 3 §3.3) обязаны блокировать переходы, нарушающие предусловие соответствующего гейта.

6.13 Связь с другими главами

Глава	Связь
02 Положение в типологии методологий	Иерархия BR / SR / TR — несущая структура инверсии источника истины (Утверждение 1); waterfall-форма (Утверждение 2) задаёт слои BR → SR → TR
07 ADAPT	`BR.source.adapt` , `SR.source.adapt` — условные (при отсутствии ADAPT — `source.adversarial-review-ref`); ось требований выводится из approved ADAPT либо напрямую из T3 при вердикте «no findings»
08 Specifications	Параллельная ось SPEC; `SR.constrained-by[]` , `TR.implements-spec[]` — типизированные рёбра графа
09 Test cases	TC верифицируют BR / SR / TR; `verified-by[]` — auto-derived inverse
10 Жизненный цикл и QG	Машины состояний BR / SR / TR; QG-0 / QG-2 для оси требований (QG-1 — только для TC)
03 Версионирование носителя	Неизменяемые ID (V1); atomic change unit при переподчинении (V2); diff & review для approve (V3); версионирование без потери истории (V4); pinning SR-version в TR (V5); нативная для носителя подпись approve с author + timestamp (V6)
11 Maturity model	RENAR-1: ось BR / SR / TR обязательна; RENAR-3+: `constrained-by[]` для всех SR, где применимо
13 Соответствие	Закрытый список типов (BR / SR / TR) — обязательное положение v1.0; закрытый список уровней (система / подсистема / модуль) — обязательное положение v1.0
reference/02 — schemas	Полная машиночитаемая schema BR / SR / TR frontmatter

07. ADAPT — двусторонняя адаптация ТЗ

7.1 Что такое ADAPT и зачем он

Клиент присылает ТЗ на своём языке — языке бизнеса и договора. Оно подписано и больше не меняется: это контракт. Но превратить его в точные требования напрямую почти никогда не выходит. Где-то ТЗ молчит о важном («экспорт данных» — в каком формате? за какой срок?), где-то противоречит само себе, где-то один и тот же термин значит у клиента и у инженера разное. Редактировать ТЗ нельзя — договор. Молча додумать за клиента тоже нельзя: так в требования просачиваются чужие домыслы, а на приёмке всплывает «мы не это заказывали».

ADAPT — мост через этот разрыв. У него две стороны: **прямая интерпретация** («раздел §4.2 ТЗ мы поняли так-то») и **обратные находки** («§4.3 не задаёт срок — уточните»), которые уходят клиенту и возвращаются ответами. Когда обе стороны согласованы и подписаны — и клиентом, и архитектором, — ADAPT застывает **по своему предмету**, и из него выводятся затронутые BR / SR / SPEC. Так интерпретация ТЗ становится явной, зафиксированной и проверяемой, а не живёт в голове исполнителя.

ADAPT не обязан предшествовать всей деривации. Типовой повод его создать — импорт ТЗ, но вопрос к клиенту с корнем в ТЗ может всплыть и **позже** — уже в ходе декомпозиции BR → SR → SPEC или при разработке тест-кейсов. Тогда возникает **новый** ADAPT, привязанный к той стадии, где находка обнаружена, а ранее выведенные артефакты остаются валидными. ADAPT создаётся не всегда: если конвертация соответствующего фрагмента ТЗ однозначна и вопросов нет, он не нужен (§7.4.1).

ADAPT — следствие Утверждения 2 из §2.4 (RENAR — waterfall-форма ≠ классический waterfall, потому что ADAPT обеспечивает двустороннюю адаптацию вместо «переброса спецификации через забор»).

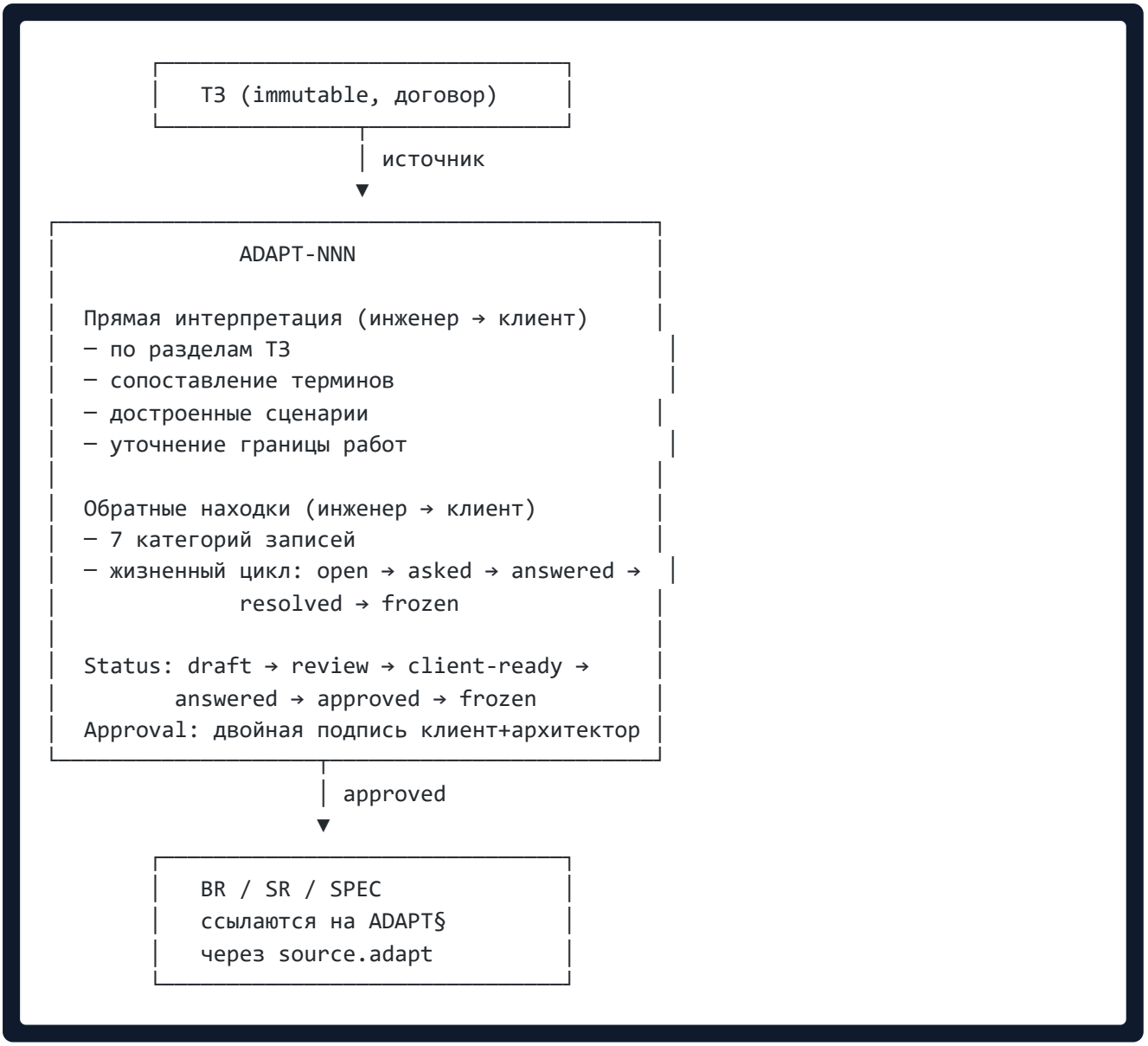
7.2 Проблема, которую нормирует ADAPT

Без формализованного промежуточного артефакта между ТЗ и BR/SR возникает один из двух негативных исходов:

Исход	Что происходит
Дрейф ТЗ	ТЗ редактируется после подписания → нарушается договор
Скрытая интерпретация	Инженерные предположения молча просачиваются в BR/SR/SPEC → разрушается цепочка прослеживаемости и происхождение

ADAPT исключает оба исхода: ТЗ остаётся неизменяемым договорным документом, **все** интерпретации, уточнения и обратные находки регистрируются в ADAPT, который сам становится неизменяемым после двойной подписи (§7.5).

7.3 Цикл двусторонней адаптации



T3 — первоисточник; ADAPT — канонический источник интерпретации; BR/SR/SPEC ссылаются на ADAPT, а не на T3 напрямую.

Показанный вход «T3 → ADAPT» — типовой (импорт T3), но **не единственный**. Триггер цикла стадийно-независим: если вопрос к клиенту с корнем в T3 всплывает позже — на стадии декомпозиции BR → SR → SPEC или при разработке TC, — цикл запускается повторно и порождает **новый** ADAPT-NNN, привязанный к стадии, на которой находка обнаружена (§7.4.1.1). На одно T3 таких ADAPT может быть несколько (MVR-3, §0.5).

7.4 Нормативные требования к ADAPT

7.4.1 Реактивная обязательность

ADAPT — **реактивный артефакт**: создаётся тогда и только тогда, когда конвертация T3 → RENAR-описание порождает разрыв между языком клиента и языком требований (§7.12). Если конвертация

однозначна — ADAPT не создаётся, BR / SR / SPEC выводятся напрямую из T3 через обязательное поле `source.tz-section`.

7.4.1.1 Условия обязательности ADAPT

ADAPT обязателен **тогда и только тогда**, когда хотя бы одно из условий выполнено:

1. **Backward finding обнаружена.** Состязательный рецензент (§7.10.2) на любой стадии жизненного цикла требований (импорт T3 либо декомпозиция BR → SR → SPEC → TC) выявил ≥ 1 запись хотя бы по одной из 7 категорий §7.4.4 (`contradiction`, `gap`, `hidden-assumption`, `feasibility`, `regulatory`, `terminology`, `scope`) с корнем в языке или намерении T3.
2. **Требуется сопоставление терминов (term mapping).** T3 использует термин, не имеющий однозначной инженерной интерпретации (требует сопоставления «клиент → инженер»).
3. **Требуется уточнение границы работ.** Граница работ из T3 неоднозначна и требует фиксации «входит / не входит».

Если ни одно условие не выполнено (состязательный рецензент выдал вердикт «no findings, no clarifications») — ADAPT **не создаётся**. BR / SR / SPEC ссылаются на T3 напрямую через `source.tz-section` (§6.5.2, §6.6.2, §8.5).

7.4.1.2 Состязательный рецензент как обязательный гейт

Состязательный обзор T3 (§7.10.2) — **обязательный шаг** для каждого T3 при импорте, независимо от того, создаётся ADAPT или нет. Вердикт при импорте — однократный, но **не финальный**: на стадиях деривации (BR → SR → SPEC → TC) состязательный рецензент выносит вердикт повторно, по мере того как декомпозиция вскрывает новые вопросы к T3. Вердикт «no findings» при импорте не блокирует появление ADAPT позже, если на стадии декомпозиции обнаружится находка с корнем в T3 (§7.7.3). Состязательный рецензент выносит формальный вердикт в одной из двух форм:

Вердикт	Что фиксируется	Следствие
«findings present»	Список конкретных findings по 7 категориям + указание разделов T3	ADAPT обязателен; жизненный цикл §7.4.5 запускается
«no findings, no clarifications»	Подтверждение состязательного рецензента (другая модель, §9.4) что T3 конвертируется в RENAR однозначно	ADAPT можно не создавать; вердикт фиксируется в свидетельствах носителя (V6 author + timestamp), доступен для аудита

Hook (§10.11.1 `adapt-applicability validation`) при создании BR / SR / SPEC без `source.adapt` проверяет наличие свидетельств вердикта «no findings» для соответствующего T3. Отсутствие свидетельств — fatal: создание блокируется до прохождения состязательного обзора.

7.4.1.3 Запрет молчаливого пропуска

Создание BR / SR / SPEC из T3 **без состязательного обзора** (пропуск ADAPT без вердикта) — нарушение стандарта. Это сохраняет инверсию источника истины (§2.3): «no findings» — это **зафиксированное утверждение** состязательного рецензента, не молчаливое допущение архитектора. Вердикт обязан быть фиксирован нативным для носителя механизмом V6 (author + timestamp) и доступен по запросу аудитора (§13.5).

При наличии delta-T3 — то же правило: delta-ADAPT создаётся реактивно, по факту находок при состязательном обзоре delta-T3 (§7.6).

7.4.1.4 Множественность ADAPT на одно ТЗ

Поскольку триггер стадийно-независим (§7.4.1.1), на одно немодифицированное ТЗ может приходиться **ноль или более** корневых ADAPT (cardinality 0..N — переформулировка MVR-3, §0.5):

- **ноль** — состязательный обзор на всех стадиях вынес «no findings»;
- **один** — находка обнаружена однократно (как правило при импорте);
- **несколько** — находки с корнем в ТЗ возникали на разных стадиях деривации (импорт, затем декомпозиция BR → SR и т. д.).

Каждый ADAPT сохраняет стабильный сквозной `ADAPT-NNN` и фиксирует поле `trigger-stage` — стадию, на которой он порождён (§7.8.1). Множественность — **штатный** случай, а не исключение, и не ослабляет провенанс: каждый BR / SR / SPEC по-прежнему ссылается ровно на один ADAPT (через `source.adapt`) либо напрямую на ТЗ (через `source.tz-section`), из которого выведен.

7.4.2 Неизменяемость ТЗ

ТЗ — договорной документ. После регистрации ТЗ в носителе его содержимое **не редактируется**. Если в ходе работы выясняется, что в ТЗ ошибка / пропуск / противоречие — это регистрируется как обратная находка в ADAPT (§7.4.4), клиент даёт ответ, ответ становится частью ADAPT. ТЗ остаётся неизменяемым.

При большом количестве правок или при изменении границы работ — клиент подписывает delta-ТЗ как новый неизменяемый документ (§7.6).

7.4.3 Прямая адаптация ТЗ

Раздел прямой интерпретации ADAPT для каждого раздела ТЗ обязан содержать:

Элемент	Обязательность	Цель
Точная ссылка на ТЗ§N.N	обязательно	происхождение
Цитата из ТЗ (или пересказ с явной пометкой)	обязательно	Контекст
Инженерная интерпретация раздела	обязательно	Перевод языка клиента → язык требований
Сопоставление терминов (клиент → инженер)	обязательно если применимо	Дезамбигуация терминов
Достроенные сценарии	обязательно если ТЗ их подразумевает	Явная фиксация неявных случаев
Уточнение границы работ (входит / не входит)	обязательно	Граница работ
Ссылки из прямой интерпретации на BR/SR/SPEC	auto-derived	Цепочка прослеживаемости (§7.7)

7.4.4 Обратные находки и их категории

Раздел обратных находок ADAPT фиксирует обнаруженные проблемы по семи нормативным категориям. **Список категорий закрыт на v1.0**; добавление новых категорий — через формальную процедуру изменения стандарта (см. глава 13). Общая политика закрытого списка и master-индекс — §1.7.5.

ID	Категория	Что регистрируется
`contradiction`	Противоречие	Внутренние противоречия в ТЗ (§A vs §B)
`gap`	Пропуск	ТЗ молчит о том, без чего невозможна реализация
`hidden-assumption`	Скрытое предположение	Предположение инженера, которое может быть неверно
`feasibility`	Реализуемость	Технически нереализуемое или непропорционально дорогое требование
`regulatory`	Регуляторика	Требование, затрагивающее законодательство / compliance
`terminology`	Терминология	Неясный термин ТЗ с несколькими возможными значениями
`scope`	Скоп	Неясная граница работ

Каждая запись об обратной находке имеет **стабильный ID** (`B-NNN`), неизменяемый после создания. ID не переиспользуется даже для отозванных записей (журнал аудита).

7.4.5 Жизненный цикл одной записи об обратной находке

```
open → asked-to-client → answered → resolved → frozen
              ↑                          │
              └──────── revised ─────────┘  (если ответ требует уточнения)
```

Статус	Что значит
`open`	Инженер записал; не отправлено клиенту
`asked-to-client`	Вопрос отправлен клиенту, ожидается ответ; зафиксирована дата
`answered`	Клиент ответил; ответ записан в документ с author + timestamp (возможность носителя V6)
`resolved`	Инженер интегрировал ответ в прямую интерпретацию
`revised`	Ответ клиента расплывчатый; повторный вопрос. Переход обратно в `asked-to-client`
`frozen`	После approval ADAPT — изменения невозможны

Approval ADAPT (§7.5) запрещён при наличии хотя бы одной записи об обратной находке в статусе `open` / `asked-to-client` / `answered` / `revised` . Все такие записи обязаны быть в

`resolved` перед approval.

7.5 Утверждение ADAPT — двойная подпись

ADAPT переходит из статуса `answered` в `approved` только после **двойной подписи** (атомарная единица изменения, возможности носителя V2 + V3 из §3.3):

Подпись	Кто	Что подтверждает
Подпись клиента	Клиент или представитель клиента с полномочиями	Прямая интерпретация ТЗ совпадает с тем, что заказчик имел в виду; ответы на вопросы по обратным находкам — финальные
Подпись архитектора	Архитектор со стороны исполнителя	Все обратные находки отработаны (нет нерешённых); прямая интерпретация технически реализуема

Нативная для носителя реализация подписи — комбинация V3 (diff & review) + V6 (author + timestamp). Конкретный механизм (digital signature / процесс утверждения / двойная review-аттестация) выбирается реализацией и фиксируется в манифесте соответствия (§3.7).

После утверждения ADAPT **неизменяем** наравне с ТЗ. Дальнейшие изменения — только добавлением нового артефакта с типизированной связью, по одному из трёх непересекающихся механизмов: delta-ADAPT при delta-ТЗ (§7.6.1), errata при ошибке интерпретации (§7.6.3) или дезавуирование ранее верного решения (§7.6.4). Сам frozen ADAPT не редактируется ни в одном из них.

7.6 Delta-ТЗ и delta-ADAPT

7.6.1 Рабочий процесс

Delta-ADAPT следует тому же реактивному правилу §7.4.1, что и корневой ADAPT: создаётся по факту находок при состязательном обзоре delta-ТЗ.

При появлении delta-ТЗ от клиента:

1. Клиент регистрирует delta-ТЗ в носителе как новый неизменяемый документ (`TZ-YYYY-NNN-delta-N`).
2. Клиент подписывает delta-ТЗ (V6 author + timestamp).
3. Состязательный рецензент (§7.10.2) проводит обзор delta-ТЗ и выносит вердикт.
4. **Если вердикт «findings present»** — Архитектор / AI-агент создаёт delta-ADAPT (`ADAPT-NNN-delta-N`) с frontmatter полем `parent-adapt: ADAPT-NNN` и `source-tz: TZ-YYYY-NNN-delta-N` . Forward интерпретация покрывает только разделы delta-ТЗ. Backward findings фиксируются по delta-ТЗ. Approval — двойная подпись §7.5.
5. **Если вердикт «no findings, no clarifications»** — delta-ADAPT не создаётся. Вердикт фиксируется в носителе с V6 author + timestamp. BR / SR / SPEC, изменяемые в результате delta-ТЗ, ссылаются на `source.tz-section: TZ-YYYY-NNN-delta-N` напрямую.

В обоих случаях свидетельство состязательного обзора (вердикт) обязано быть машинно-доступно для аудита (hook §10.11.1 reference-validation).

7.6.1bis Пример: тривиальный delta-T3

Delta-T3: «переименовать поле "username" на форме регистрации в "email"».

1. Клиент подписывает `TZ-YYYY-NNN-delta-1`.
2. Состязательный рецензент проверяет: термин однозначен (`email` — стандартный inженерный термин), scope чёткий (одно поле на одной форме), вопросов к клиенту нет.
3. Вердикт: «no findings, no clarifications»; зафиксирован в носителе.
4. delta-ADAPT **не создаётся**.
5. SR-NN с поведением «POST /auth/sign-up принимает email» получает обновление: `source.tz-section: TZ-YYYY-NNN-delta-1 §1`. SR `parent` BR-NN не меняется.

7.6.2 Цепочка delta-ADAPT

```
ADAPT-001 (от TZ-YYYY-NNN main)
  └─ ADAPT-001-delta-1 (от TZ-YYYY-NNN-delta-1)
       └─ ADAPT-001-delta-2 (от TZ-YYYY-NNN-delta-2)
            └─ ADAPT-001-delta-3 (от TZ-YYYY-NNN-delta-3)
```

Цепочка строго последовательна: применение delta-ADAPT обязано идти по порядку (см. сквозную фиксацию версии V5 в §3.3.5). Перенумеровать или переставлять delta-ADAPT в цепочке запрещено.

7.6.3 Errata для уже approved ADAPT

Если через продолжительное время после утверждения обнаруживается, что в ADAPT-NNN неверна прямая интерпретация T3 или некорректно зафиксировано разрешение обратной находки — два допустимых исхода:

Исход	Артефакт
T3 содержит неоднозначность, обнаруженную поздно	delta-ADAPT с новой записью об обратной находке и ответом клиента
Неверная интерпретация (ошибка инженера)	errata-ADAPT-NNN-M как отдельный артефакт с подписью клиента (если меняет контрактный итог) или только архитектора (если правка косметическая)

В обоих исходах **frozen ADAPT не редактируется**. Только добавление новых артефактов с явной типизированной связью.

7.6.4 Дезавуирование approved ADAPT

Delta и errata не покрывают один случай: ранее принятое решение было **верным**, но позже сформированные требования ему **противоречат** по новому пониманию. Это не ошибка инженера (errata) и не новый контракт от клиента (delta) — это отмена ранее корректного решения. Для него вводится третий механизм — **дезавуирование** (`supersession`).

Механизм	Когда	Природа

delta-ADAPT (§7.6.1)	пришло delta-T3 от клиента	новый источник: изменился контракт
errata-ADAPT (§7.6.3)	прежняя интерпретация была **ошибочной**	исправление того, что всегда было неверно
дезавуирующий ADAPT (§7.6.4)	прежнее решение было **верным**, но позже сформированные требования ему противоречат	отмена ранее корректного решения по новому пониманию

Правила дезавуирования:

1. **Дезавуирующий артефакт.** Создаётся новый `ADAPT-NNN` с frontmatter-полем `supersedes: ADAPT-MMM` и обязательным `supersession-rationale`, ссылающимся на конкретное противоречащее требование (`BR` / `SR` / `SPEC` ID) и его источник. Дезавуируемый ADAPT получает автоматически выводимую обратную ссылку `superseded-by: ADAPT-NNN` (§7.8.1).
2. **Подпись.** Если дезавуируемое решение имело **контрактный итог** (было подписано клиентом — типовой случай для approved ADAPT) — дезавуирование требует **новой подписи клиента**: согласованное с клиентом решение нельзя отменить в одностороннем порядке. Если же правка строго косметическая и не затрагивает контрактный итог, допустима подпись только архитектора, без клиента (по аналогии с правилом errata, §7.6.3). Отдельный QG не вводится — дезавуирование проходит через тот же QG-3 (двойная подпись, §7.5, §10.8.5).
3. **Состояние.** Дезавуируемый `ADAPT-MMM` переходит в выделенное терминальное состояние `superseded` — отдельное от `obsolete` (устаревание) и от `frozen`. `superseded` immutable и **сохраняется** для аудита (immutable history, возможность носителя V1); не удаляется. Переход нормирован в §10.8.5.
4. **Перенаправление производных.** Все `BR` / `SR` / `SPEC` с `source.adapt: ADAPT-MMM` обязаны быть либо перенаправлены на дезавуирующий ADAPT, либо пере-выведены. Висячая ссылка `source.adapt` на ADAPT в статусе `superseded` — **fatal**: гейт `check-adapt-supersession.js` блокирует (§10.11.1), по аналогии с reference-validation.
5. **Аддитивность.** Как delta и errata — дезавуируемый ADAPT **не редактируется**; вводится только новый артефакт и типизированная связь `supersedes` / `superseded-by`.

Дезавуирование — расширяющая возможность: одиночный ADAPT без дезавуирований остаётся валидным частным случаем, миграция существующих проектов не требуется.

7.7 Связь ADAPT с другими артефактами

7.7.1 BR / SR / SPEC ссылаются на ADAPT через `source.adapt`

```yaml
# Frontmatter SR (пример)
source:
  adapt: ADAPT-001
  adapt-section: "Forward §3"   # прямая интерпретация; канонический
```

```
идентификатор секции — Forward §*
  tz-section: "§3.4"            # для traceability; первоисточник остаётся ТЗ
```

Поле `source.adapt` — условное: присутствует, когда конвертация ТЗ → RENAR потребовала ADAPT; опускается, когда состязательный рецензент вынес вердикт «no findings» — тогда обязательно поле `source.adversarial-review-ref` (§7.4.1). Поле `source.tz-section` — обязательное всегда (двойная цепочка прослеживаемости).

7.7.2 Полная цепочка прослеживаемости

```
TC-NN  →  verifies SR-12  →  выведено из ADAPT-001 §4 (прямая интерпретация)
                                  │     │
                                  │     └─ resolves B-007 (was: contradiction,
                                  │              answered by client 2026-03-15)
                                  │
                                  └─ interprets TZ-YYYY-NNN §3.4
```

При проверке от тест-кейса можно дойти до: (a) исходного раздела ТЗ, (b) прямой интерпретации этого раздела, (c) вопросов исполнителя по обратным находкам, (d) ответов клиента, (e) производных BR / SR / SPEC.

7.7.3 TR не ссылается на ADAPT напрямую

TR (задача) ссылается на SR / SPEC, а они уже ссылаются на ADAPT. Исполнитель задачи **не обращается к ADAPT напрямую** — все нужные интерпретации уже содержатся в SR / SPEC. Если исполнитель обнаружил неясность в SR — корень определяет исход (Q1, §7.4.1.1):

- **корень в декомпозиции** (а не в ТЗ — например, неточная формулировка SR, упущенная связь `constrained-by[]`) — решается уточнением SR / SPEC **без** ADAPT;
- **корень в языке или намерении ТЗ** — регистрируется обратная находка в ADAPT. Если ADAPT для этого ТЗ уже существует — добавляется новая запись; если ADAPT ещё не создавался (вердикт при импорте был «no findings») — на текущей стадии создаётся **новый** ADAPT (§7.4.1.1). Это штатный, а не исключительный случай.

7.8 ADAPT schema

7.8.1 frontmatter (обязательные поля)

```
---
id: ADAPT-NNN                       # immutable; NNN sequential per project
title: "Адаптация ТЗ <name>"
type: ADAPT
trigger-stage: import-tz            # стадия, породившая ADAPT (§7.4.1.4):
                                    # import-tz | decompose-br | decompose-sr |
```

```
spec | tc

source-tz:
  id: TZ-YYYY-NNN
  signed-date: "<ISO-date>"
  signed-by-client: "<name + role>"
  document-version-ref: "<нативный для носителя идентификатор версии>"   # V5 pin
(см. §3.3.5)

parent-adapt:                       # для delta-ADAPT
  id: ADAPT-NNN
  delta-tz: TZ-YYYY-NNN-delta-N

supersedes: ADAPT-MMM               # только для дезавуирующего ADAPT (§7.6.4);
опускается иначе
superseded-by: ADAPT-NNN            # auto-derived; проставляется на
дезавуируемом ADAPT
supersession-rationale: >           # mandatory если присутствует supersedes:
  "<ссылка на противоречащее BR/SR/SPEC ID + его источник; обоснование отмены>"

status: draft | review | client-ready | answered | approved | frozen | superseded
| obsolete
created: "<ISO-date>"
last-updated: "<ISO-date>"

approval:                           # обязательно для approved
  client-signature:                 # mandatory для approved
    signed-by: "<name>"
    role: "<role>"
    organization: "<client-org>"
    signed-at: "<ISO-datetime>"     # V6 timestamp
    signature-ref: "<нативная для носителя ссылка>"
  architect-signature:              # mandatory для approved
    signed-by: "<name>"
    role: architect
    signed-at: "<ISO-datetime>"

generates-requirements: []          # auto-derived; BR/SR из этого ADAPT
generates-specs: []                 # auto-derived; SPEC-* из этого ADAPT
open-questions-count: integer       # auto-derived; обязательно 0 для approved
resolved-questions-count: integer

ai-provenance:                      # обязательно если ADAPT draft был AI-
сгенерирован
  generated-by: "<vendor>-<model>@<date>"
  prompt-template: "<template-path>@<version>"
  context-tokens: integer
  output-tokens: integer
  human-edits: boolean              # обязательно true для approved — клиент
видел текст
---
```

Примечание: ADAPT существует только в одном режиме (двойная подпись, полный жизненный цикл §7.4.5). Реактивность (§7.4.1) выражается **на уровне создания**: если состязательный рецензент вынес вердикт «no findings, no clarifications», ADAPT не создаётся вовсе, а не создаётся в упрощённой форме.

7.8.2 Body structure (обязательные разделы)

Обязательные разделы тела ADAPT:

1. **Краткое содержание** — 3–5 параграфов для одностраничного чтения клиентом.
2. **Сопоставление терминов (Term mapping)** — таблица «клиент → инженер».
3. **Прямая интерпретация (раздел Forward)** — раздел на каждый раздел ТЗ с обязательными элементами из §7.4.3.
4. **Обратные находки** — все записи с жизненным циклом из §7.4.5.
5. **Резюме по обратным находкам** — статистическая таблица по категориям и статусам.
6. **Таблица производных артефактов** — auto-derived список BR / SR / SPEC.
7. **История изменений ADAPT** — нативно для носителя, auto-generated.

Опциональные разделы — на усмотрение реализации, не нормированы.

7.9 Контрольные точки качества для ADAPT

ADAPT имеет выделенную машину состояний. Детали в главе 10 §10.4. Краткая сводка:

Gate	Предусловие	Постусловие
QG-ADAPT-draft	ADAPT создан; присутствует frontmatter	Прямая интерпретация охватывает все разделы ТЗ
QG-ADAPT-review	Прямая интерпретация заполнена; первичные обратные находки в `open`	Все обратные находки в `open` или `asked-to-client`
QG-ADAPT-client-ready	Все обратные находки в `asked-to-client`; пакет вопросов сформирован	Готов к отправке клиенту
QG-ADAPT-answered	Все обратные находки в `answered`; начато разрешение	Готов к финализации
QG-ADAPT-approve	Все обратные находки в `resolved`; двойная подпись готова	ADAPT неизменяем; разрешена генерация BR / SR / SPEC
QG-ADAPT-frozen	`approved`	Дальнейшие изменения — только через delta-ADAPT или errata

Hooks носителя (§3.3.3 V3, §3.3.5 V5) обязаны:

- Блокировать переход в `approved` при `open-questions-count > 0`.
- Блокировать создание BR / SR / SPEC с `source.adapt` на ADAPT в статусе ниже `approved`.
- Пересчитывать `open-questions-count` / `resolved-questions-count` после каждого изменения.

7.10 ADAPT и AI-генерация

7.10.1 AI-агент создаёт draft ADAPT

При импорте ТЗ AI-агент создаёт **draft ADAPT** автоматически: прямая интерпретация по разделам, попытка обнаружить contradictions / gaps / terminology ambiguities, первая версия сопоставления терминов. Этот draft — стартовая точка для архитектора, не финальный артефакт.

7.10.2 Состязательный рецензент

Применение **принципа состязательного обзора** (отдельный AI-агент-критик с **другой моделью**; процедура — guide/07 §4.5): ищет, что primary агент пропустил — недостающие обратные находки. Если состязательный рецензент находит не менее трёх серьёзных новых находок — primary draft возвращается на доработку.

7.10.3 Клиент не общается с AI напрямую

Вопросы по обратным находкам агрегируются архитектором в человеческий формат **перед отправкой клиенту**. Архитектор может убрать дубликаты, переформулировать на язык клиента, объединить связанные. Клиент видит подготовленный список вопросов, не raw output AI-агента. Ответ клиента (в любой форме: текст, видеозвонок, письмо) транскрибируется архитектором в ADAPT с указанием канала и аутентификации.

7.11 Схема хранения

ADAPT-документы хранятся в подпапке `adapt/` носителя требований:

```
[project]/
  adapt/
    ADAPT-001-main.md
    ADAPT-001-delta-1.md
    ADAPT-001-delta-2.md
    ADAPT-002-main.md
    errata/
      errata-ADAPT-001-1.md
  tz/
    TZ-YYYY-NNN.md
    TZ-YYYY-NNN-delta-1.md
```

Конкретная файловая структура на конкретном носителе специфична для носителя (см. guide/03 для distributed VCS; guide/04 для document-oriented store).

7.12 Соотношение ТЗ и RENAR-описания

7.12.1 Два языка с переменным расстоянием

ТЗ и RENAR-описание — два разных артефакта с разной природой:

Параметр	ТЗ	RENAR-описание
Целевой читатель	Клиент (человек)	AI-агент (§0.2.1) и человек-verifier
Язык	Язык клиента (бизнес-домен, контрактная лексика)	Язык требований (канонические IDs, закрытые списки, формальные frontmatter и графовые связи)
Полнота	Допускает умолчания, неполноту, неоднозначность формулировок	Не допускает умолчаний; полнота — нижняя граница machine-enforceability (§0.3)
Эволюция	Инкрементная: первое ТЗ описывает систему полностью, последующие — только delta (§7.6)	Не инкрементная: перед изменением системы создаётся новая полная версия RENAR-описания, и только затем AI-агент производит изменения в реализации
Статус после подписания	Неизменяемый договорной документ (§7.4.2)	Эволюционирует через approved delta-ADAPT или (когда ADAPT не создаётся, §7.4.1) через source.tz-section напрямую

Расстояние между двумя языками **переменное**. Иногда разделы ТЗ формулированы достаточно однозначно, чтобы AI-агент конвертировал их в BR/SR/SPEC без потерь смысла. Иногда — наоборот: ТЗ содержит контрактную фразу, имеющую несколько инженерных трактовок, или пропускает поведение, без которого реализация невозможна.

7.12.2 ADAPT — реактивный мост между языками

ADAPT существует только когда между языками возникает разрыв. Его прямая интерпретация (forward, §7.4.3) — **перевод** конкретных разделов ТЗ с языка клиента на язык требований. Обратные находки (backward, §7.4.4) — формализация того, что при переводе обнаружились пробелы, неоднозначности или противоречия, требующие согласования с клиентом.

Из этой природы следуют три следствия:

1. **ADAPT — реактивный артефакт по дизайну.** Существование разрыва между языками — необходимое условие создания (§7.4.1.1); формальный ADAPT без содержания утрачивает смысл для аудита и обесценивает остальные ADAPT.
2. **Состязательный рецензент — единственный, кто декларирует отсутствие разрыва.** «ADAPT не нужен» — зафиксированный вердикт другой модели (§7.10.2, §7.4.1.3), а не молчаливое допущение архитектора.
3. **Форма ADAPT — единственная.** Создаётся в полной форме (двойная подпись §7.5, все разделы body §7.8.2, полный жизненный цикл §7.4.5); промежуточных «light»-форм не существует.

7.12.3 Почему RENAR-описание всегда полное

RENAR-описание не инкрементное по дизайну: AI-агент (§0.2.1) не способен надёжно «достроить» изменения, опираясь только на delta. Полная новая версия RENAR-описания — единственная форма, гарантирующая что:

- machine-enforceable инварианты (полнота, graph consistency, состояния жизненного цикла) применимы к описанию **в целом**, а не к diff;
- состязательный рецензент работает на полном артефакте, не на дельте;
- последующие шаги (декомпозиция, генерация TC, реализация — `guide/00-quickstart §«Два штатных сценария»`) выполняются на актуальной полной картине системы.

Delta-T3 — инкрементное на уровне **источника** (контрактная сторона); полная новая версия RENAR-описания собирается AI-агентом из родительского RENAR + либо delta-ADAPT (если он создавался), либо напрямую с учётом delta-T3 через `source.tz-section`.

7.12.4 Связь с инверсией источника истины

T3 — договорной артефакт, но **не** источник истины о поведении системы (§2.3). Источник истины — RENAR-описание (BR / SR / SPEC / TR / TC). T3 — источник, из которого RENAR-описание выводится **либо** через ADAPT (когда есть разрыв), **либо** напрямую через `source.tz-section` (когда разрыв отсутствует); код — производный артефакт реализации RENAR-описания. ADAPT и §7.12 фиксируют двойную инверсию: контрактный источник (T3) → источник истины (RENAR-описание) → код. ADAPT встраивается в цепочку реактивно, когда мост между языками нужен.

7.13 Связь с другими главами

Глава	Связь
02 Methodology positioning	ADAPT — следствие Утверждения 2 (двусторонняя адаптация вместо «переброса спецификации через забор»)
06 Requirements hierarchy	BR / SR ссылаются на ADAPT через `source.adapt`
08 Specifications	SPEC-* также ссылаются на ADAPT через `source.adapt`
09 Test cases	TC через SR / SPEC возвращается в ADAPT для полной цепочки прослеживаемости
10 Жизненный цикл и QG	машина состояний ADAPT + QG-ADAPT-* gates
03 Версионирование носителя	Двойная подпись (V6) + atomic approval (V2 + V3); immutability после approved (V1); delta-ADAPT через V4
13 Соответствие	ADAPT для каждого T3 — обязательное положение

08. Спецификации — 9 типов SPEC

> ***Часть RENAR Standard v1.0-draft*** · *← Оглавление*

8.1 Зачем отдельная ось спецификаций

Возьмём требование «система создаёт заказ». Оно говорит, **что** должно происходить — но молчит о том, **как** система для этого устроена: какой у неё API-контракт, в какой таблице лежит заказ, по каким правилам доступа, на каком экране. Втиснуть всё это в само требование не выйдет — оно превратится в кашу. Поэтому RENAR разводит описание на две оси: **поведение** (BR / SR / TR, глава 6) и **структуру** — спецификации, SPEC.

Спецификация — не «более детальный SR» и не его ребёнок. Одно требование «создать заказ» обычно опирается сразу на пять-семь спецификаций (архитектура, API, данные, процесс, безопасность, экран), поэтому связь между осями — граф типизированных рёбер (`constrained-by[]` , `implements-spec[]`), а не дерево. Типов спецификаций ровно девять, и список закрыт: `SPEC-ARCH` , `API` , `DATA` , `INT` , `PROC` , `UI` , `AI` , `SEC` , `OPS` — новый тип вводится только через формальную процедуру изменения стандарта (глава 13).

8.2 Архитектурное решение: SPEC — параллельная ось, не дети SR

8.2.1 Две оси описания системы

Требования и спецификации отвечают на разные вопросы:

Ось	Артефакты	Вопрос
Поведенческая	BR / SR / TR (глава 6)	Что система должна делать
Структурная	SPEC-* (9 типов)	Как система структурно устроена для выполнения этих требований

8.2.2 SPEC как параллельная ось: связи через типизированный граф

Связи между осью требований (BR / SR / TR) и осью SPEC организованы как **граф зависимостей**, не дерево родителей. У SR ровно один родитель в дереве требований (BR), но множество типизированных рёбер `constrained-by[]` на SPEC. Один SR обязан ссылаться на каждую SPEC, ограничивающую его поведение в осях API / data / UI / process / security / ops; обратно — одна SPEC может ограничивать множество SR.

Пример: SR «создать заказ» опирается на SPEC-ARCH (где живёт компонент заказов), SPEC-API (контракт endpoint), SPEC-DATA (схема таблицы), SPEC-PROC (workflow), SPEC-SEC (правила доступа), SPEC-UI (форма).

Нормативно: каждое ребро `SR.constrained-by[]` и `TR.implements-spec[]` **должно** ссылаться на одну из закрытых категорий SPEC, перечисленных в §8.3; ad-hoc категории не допускаются (см. §1.7 closed-list policy).

```
Дерево требований (поведенческая ось):     Параллельная ось спецификаций:

BR                                           SPEC-ARCH    SPEC-API
 └── SR  ◄──── constrained-by[] ────►        SPEC-DATA    SPEC-INT
      └── TR ─── implements-spec[] ──►       SPEC-PROC    SPEC-UI
                                             SPEC-AI      SPEC-SEC
                                             SPEC-OPS

Дерево требований:
  SR.parent              → BR              (единственный родитель)
  TR.parent              → SR              (единственный родитель)

Граф связей (типизированные рёбра):
  SR.constrained-by[]    → SPEC-*
  TR.implements-spec[]   → SPEC-*
  SPEC-*.depends-on[]    → SPEC-*          (между спецификациями)
  SPEC-*.referenced-by[] → SR / TR          (auto-derived inverse)
```

8.2.3 Обоснование

Аргумент	Следствие
SPEC и SR отвечают на разные вопросы	SPEC не уточняет SR на более глубоком уровне — это отдельная категория описания
Один SR опирается на 5–7 SPEC	Дерево «SPEC как родитель SR» приводит к множественному родительству
Industry standards (arc42, C4, OpenAPI, BPMN, ERD) живут параллельно требованиям	RENAR следует этой проверенной практике
AI-агент может параллелить генерацию SR и SPEC	Без блокировки одного типа другим

8.3 Закрытый список из девяти типов SPEC

Тип	Назначение	Industry reference
`SPEC-ARCH`	Архитектура системы / подсистемы: контексты, контейнеры, компоненты, deployment view, quality attributes	arc42, C4 model (Brown), ISO/IEC/IEEE 42010
`SPEC-API`	API contracts: REST / GraphQL / gRPC / async events; версионирование, error model, rate limits	OpenAPI 3.x, AsyncAPI 2.x, gRPC IDL
`SPEC-DATA`	Модель данных: schema, ERD, indices, миграции, retention, PII classification	ISO/IEC 11179, JSON Schema

`SPEC-INT`	Integration: взаимодействие между подсистемами и внешними системами; протоколы, контракты, SLA	Enterprise Integration Patterns (Hohpe)
`SPEC-PROC`	Process / workflow: бизнес-процессы, state machines, saga, choreography, orchestration	BPMN 2.0, ISO/IEC 19510
`SPEC-UI`	UI / UX: экраны, навигация, user journeys, accessibility, i18n, эталонные изображения	Material Design / Apple HIG, WCAG 2.2
`SPEC-AI`	AI / ML: model cards, RAG, prompt engineering, eval strategy, cost budget	ISO/IEC 23894, NIST AI RMF
`SPEC-SEC`	Security: authn / authz, threat model, secrets management, data classification	STRIDE, OWASP ASVS, ISO/IEC 27001
`SPEC-OPS`	Operations: deployment, observability, SLO / SLA, runbook, disaster recovery	Google SRE, ITIL v4, ISO/IEC 20000

8.3.1 Что НЕ вошло в v1.0 (с обоснованием)

Кандидат	Решение	Обоснование
`SPEC-EVENT`	Не отдельный тип	События / очереди — раздел SPEC-API (asynchronous APIs)
`SPEC-CONFIG`	Не отдельный тип	Feature flags / env vars / secrets — раздел SPEC-OPS
`SPEC-PERF`	Не отдельный тип	Performance / NFR — раздел SPEC-ARCH (quality attributes) или SPEC-OPS (SLO)
`SPEC-TEST-ENV`	Не отдельный тип	Тестовые окружения — раздел SPEC-OPS
`SPEC-DOMAIN`	Не отдельный тип	Domain model — поглощён SPEC-ARCH (decomposition) + SPEC-DATA (entities)
`SPEC-MIGRATION`	Не отдельный тип	Migration — раздел SPEC-DATA (жизненный цикл)
`SPEC-COMPLIANCE`	Не отдельный тип	Compliance — связи между SR/SPEC и нормативами через `compliance-refs[]`, не отдельный артефакт

Политика закрытого списка: если в дальнейшей работе обнаружится, что какой-то из исключённых типов реально нужен — он добавляется через формальную процедуру изменения стандарта с обоснованием.

8.4 Общая схема (общие поля frontmatter)

Все 9 типов SPEC делят общий набор frontmatter полей. Поля, специфичные для типа, добавляются как расширения поверх (§8.5). Полная machine-readable модель данных — в reference/02-schemas.md.

```yaml
---
# === Identity (обязательно) ===
id: SPEC-<TYPE>-NN[.N]              # immutable; TYPE ∈
{ARCH,API,DATA,INT,PROC,UI,AI,SEC,OPS}
title: "<short, descriptive>"
type: SPEC-ARCH | SPEC-API | SPEC-DATA | SPEC-INT | SPEC-PROC | SPEC-UI | SPEC-AI
| SPEC-SEC | SPEC-OPS
slug: "<kebab-case>"               # auto-derived

# === Scope (обязательно) ===
level: system | subsystem | module
scope:
  system: "<system-id>"
  subsystem: "<subsystem-id>"       # null если level=system

# === Жизненный цикл (обязательно) ===
status: draft | review | approved | verified | obsolete
priority: must | should | could     # not all types use; mostly SPEC-SEC / SPEC-
OPS

# === Source: provenance (conditional, см. глава 7 §7.4.1) ===
# source.adapt — conditional (present когда ADAPT создавался; §7.4.1.1).
# source.tz-section — обязательно всегда.
# source.adversarial-review-ref — обязательно когда source.adapt omitted.
source:
  adapt: ADAPT-NNN                  # conditional
  adapt-section: "Forward §N"       # обязательно если adapt present
  tz-section: "§N.N"                # обязательно всегда
  adversarial-review-ref: "<нативная для носителя ссылка>"   # обязательно если
adapt omitted

# === Граф связей (auto-managed except mandatory ones) ===
referenced-by: []                   # auto-derived; SR/TR/SPEC ссылающиеся сюда
depends-on: []                      # обязательно если есть; SPEC-* на которые
опирается этот SPEC
verified-by: []                     # auto-derived; список TC IDs верифицирующих

# === AI provenance (обязательно на RENAR-4+; каноническая schema — §4.10.1) ===
ai-provenance:
  generated-by: "<vendor>-<model>@<date>"
  generated-at: "<ISO-8601>"
  prompt-template: "<template-path>@<version>"
  context-tokens: integer
  output-tokens: integer
  human-edits: boolean
  generation-time-ms: integer        # optional; см. §4.10.1
  # optional на RENAR-4, обязательно на RENAR-5:
  # cost-budget, cost-actual

# === Замена (обязательно если применимо) ===
replaces: "<old-id>"
replaced-by: "<new-id>"
deprecated-date: "<ISO date>"
```

```
# === Compliance (optional) ===
compliance-refs: []                # ссылки на ISO/GDPR/AI Act/NIST AI RMF
---
```

8.4.1 Обязательные разделы body

Body любого SPEC обязательно содержит:

1. **Назначение** — 1–3 параграфа.
2. **Scope** — что входит, что не входит.
3. **Разделы, специфичные для типа** — см. §8.5.
4. **Связь с требованиями** — какие SR/BR ссылаются.
5. **Связь с другими SPEC** — `depends-on[]`.
6. **Verification** — какие TC верифицируют этот SPEC.

8.5 Расширения схемы по типам SPEC

Краткое описание специфичных для типа полей и обязательных body-разделов. Полная machine-readable extension schema — в reference/02-schemas.md. Industry references детально — в указанных стандартах.

8.5.1 SPEC-ARCH

Type-specific frontmatter: `arch-style`, `deployment-model`, `tech-stack`, `quality-attributes`.

Обязательное тело: системный контекст (C4 L1), контейнеры (C4 L2), компоненты (C4 L3) для критических контейнеров, quality attributes (latency / throughput / availability), ADR-журнал.

Spec-specific TC (глава 9): тесты соответствия архитектуры (zoning), эталонные тесты атрибутов качества.

8.5.2 SPEC-API

Type-specific frontmatter: `api-style` (rest / graphql / grpc / async-events), `api-version`, `versioning-strategy`, `authentication`, `rate-limits`, `contract-file` (location of machine-readable contract).

Обязательное тело: endpoints / operations с payload / response / errors, versioning rules (breaking vs non-breaking), error model, authn/authz reference на SPEC-SEC, rate limits, 2–3 example запросов на endpoint.

Spec-specific TC: contract tests, authentication negative, rate limit tests.

8.5.3 SPEC-DATA

Type-specific frontmatter: `data-style` (relational / document / graph / columnar), `storage-engine`, `schema-version`, `pii-classification[]`, `retention-policies[]`, `migration-strategy`.

Обязательное тело: domain entities, ERD (text / Mermaid / link), поля сущностей (type / constraints / indices / defaults), связи (FK / cardinality / cascade), PII / sensitive data classification + encryption at-rest + retention, migration approach, index strategy.

Spec-specific TC: migration tests, constraint tests (FK / NOT NULL / unique), PII handling tests, data retention tests.

8.5.4 SPEC-INT

Type-specific frontmatter: `integration-pattern` (request-response / event-driven / message-queue / webhook / file-transfer), `direction`, `counterparty`, `sla`, `idempotency`.

Обязательное тело: интегрируемые системы, контракт обмена, failure modes + retry strategy, идемпотентность + dedup, security между системами, observability (correlation IDs).

Spec-specific TC: contract tests с моком counterparty, failure injection, idempotency, end-to-end TC `tc-type: contract`.

Замечание: SPEC-INT заменяет существующий `INT-SR` (§8.7 migration).

8.5.5 SPEC-PROC

Type-specific frontmatter: `process-style` (bpmn / state-machine / saga / choreography / orchestration), `state-count`, `participants[]`, `sla` end-to-end и по шагам, `compensation` (defined / not-applicable / manual).

Обязательное тело: process diagram (BPMN-flavor / Mermaid / link), состояния и переходы (для машины состояний), participants и их роли, happy path, альтернативные сценарии и исключения, timeouts и compensation (для saga), SLA.

Spec-specific TC: happy path E2E, alternative paths, compensation tests (для saga), SLA tests.

8.5.6 SPEC-UI

Type-specific frontmatter: `ui-platform`, `target-users[]` (с ссылками на persona разделы ADAPT), `design-system`, `accessibility-level` (WCAG-A / AA / AAA), `i18n`, `mockup-links[]`, `baseline-images[]` для VLM-judge тестов.

Обязательное тело: общая структура интерфейса, ключевые экраны, user journeys без технических деталей, сквозные элементы (права доступа / уведомления / error / empty states), тон и стиль, accessibility, i18n.

Spec-specific TC: VLM-judge с эталоном (judge ≠ production изоляция), accessibility (axe-core / Pa11y), i18n (string overflow / RTL), user journey E2E.

Замечание: SPEC-UI заменяет существующий `UIC` (§8.7 migration).

8.5.7 SPEC-AI

Type-specific frontmatter: `ai-pattern` (rag / fine-tuning / prompt-engineering / tool-use / multi-agent), `production-model` (vendor / model / version), `judge-model` (обязан отличаться от production), `context-strategy`, `eval-strategy` (metric / threshold / baseline-dataset), `cost-budget`.

Обязательное тело: архитектура AI-компонент (pipeline / orchestration / fallback), model card (capabilities / limits / known failure modes), context strategy, eval strategy с изоляцией judge ≠ production, cost management, hallucination mitigation, состязательные аспекты.

Spec-specific TC: eval против эталона (judge isolated), состязательные (prompt injection как negative TC), cost regression, hallucination tests.

Замечание: SPEC-AI заменяет существующий `AIC`. Изоляция judge ≠ production model — обязательное требование стандарта для всех eval-TC.

8.5.8 SPEC-SEC

Type-specific frontmatter: `security-domains[]`, `auth-model` (authn / authz strategies), `data-classification[]` (PII-high / PCI / internal), `threat-model-method` (STRIDE / PASTA / OCTAVE), `compliance[]`, `incident-response` reference в SPEC-OPS.

Обязательное тело: auth model (authn flow / authz rules), data classification с защитой, threat model (STRIDE-таблица с mitigation на каждую угрозу), secrets management, audit (что логируется / retention / доступ), encryption (at-rest / in-transit / key management), compliance mapping (ссылки на конкретные пункты).

Spec-specific TC: authn (pos + neg), authz (RBAC matrix), threat-test (каждая STRIDE-угроза → минимум 1 negative TC), журнал аудита, secrets leakage.

8.5.9 SPEC-OPS

Type-specific frontmatter: `deployment-style`, `environments[]` (dev / staging / prod с purpose и scale), `slo`, `observability` (logs / metrics / traces / alerting), `runbook-link`, `disaster-recovery` (rto / rpo / backup-strategy).

Обязательное тело: environments, deployment process (CI/CD pipeline / gating / rollout strategy), SLO (availability / latency / error budget), observability, alerting (критические алерты / escalation), runbook, capacity planning, disaster recovery.

Spec-specific TC: deployment tests (smoke), SLO regression (load testing), failover (DR drills), observability (alerts срабатывают когда ожидается).

8.6 Связь с требованиями и задачами

8.6.1 SR.constrained-by[]

SR в frontmatter получает поле `constrained-by[]` — типизированные ссылки на SPEC. Это **граф**, не дерево родителей. Родитель SR в дереве требований — единственный (BR).

```yaml
# Frontmatter SR (пример)
id: SR-05
parent:
  id: BR-02
constrained-by:
  - SPEC-UI-01
```

```
  - SPEC-API-02
  - SPEC-DATA-03
  - SPEC-PROC-01
  - SPEC-SEC-01
verified-by:
  - TC-12
  - TC-13
source:
  adapt: ADAPT-001
  adapt-section: "Forward §3"       # см. канонический идентификатор §8.4
```

8.6.2 TR.implements-spec[]

TR (задача) ссылается на SR (родитель в дереве) + одна или более SPEC через `implements-spec[]`:

```
id: TR-42
title: "Реализовать endpoint POST /orders"
parent:
  id: SR-05
implements-spec:
  - SPEC-API-02
  - SPEC-DATA-03
verified-by:
  - TC-14
```

8.6.3 SPEC.depends-on[]

SPEC может опираться на другой SPEC:

```
id: SPEC-API-02
title: "REST API заказов"
type: SPEC-API
depends-on:
  - SPEC-DATA-03         # стабильная схема данных
  - SPEC-SEC-01          # auth model для endpoints
```

При изменении upstream SPEC (например SPEC-DATA-03) все downstream (SPEC-API-02 и через него связанные SR) обязаны быть пересмотрены: либо `verified` подтверждается (изменение совместимо), либо downstream-артефакт проходит повторную проверку по своей машине состояний (§10.7) и до её завершения не считается `verified` относительно новой версии upstream.

8.6.4 Auto-derived обратные рёбра

`SPEC.referenced-by[]` пересчитывается хуком носителя после каждого изменения SR / TR / SPEC. Orphan SPEC (без `referenced-by[]` и без активного status) — предупреждение в отчёте

качества.

8.7 Миграция UIC / AIC / INT-SR / TS → SPEC-*

8.7.1 Mapping таблица

Старый тип	Новый тип	Тип миграции
`UIC-NN`	`SPEC-UI-NN`	Переименование ID + перенос в `specs/ui/`
`AIC-NN`	`SPEC-AI-NN`	Переименование ID + перенос в `specs/ai/`
`INT-SR-NN`	`SPEC-INT-NN`	Переименование ID + перенос в `specs/int/`
`TS-NN`	`SPEC-<TYPE>-NN` (распределение)	Manual review каждого TS; AI-агент классифицирует содержимое, архитектор утверждает в один клик

8.7.2 Атомарная миграция

Миграция — одна atomic change unit (V2) на уровне проекта. Параллельное существование старых типов (UIC / AIC / INT-SR / TS) и SPEC-* как источника истины запрещено.

Процедура (независимо от носителя):

1. Подготовка: AI-агент классифицирует каждый существующий TS-NN в один из 9 типов SPEC.
2. Архитектор утверждает классификацию.
3. Atomic change: переименование IDs (UIC→SPEC-UI; AIC→SPEC-AI; INT-SR→SPEC-INT; TS→SPEC-*), перенос файлов в `specs/<type>/`, обновление всех ссылок в BR / SR / TR / TC frontmatter (`parent: UIC-NN` → `constrained-by: [SPEC-UI-NN]`).
4. Регенерация auto-derived файлов (REQUIREMENTS.md, SPECS.md, обратные рёбра).
5. CI-проверка: отсутствие orphan ссылок и старых IDs.

8.7.3 ID immutability

После миграции SPEC ID **неизменяемы** (см. V1, §3.3.1). Переименование `SPEC-API-02` → `SPEC-API-08` запрещено. Замена — через `deprecated` + новый ID с `replaces[]` .

8.8 Контрольные точки качества для SPEC

SPEC имеет выделенную машину состояний (глава 10 §10.3):

State	Условие перехода
`draft`	Создан; обязательные поля frontmatter заполняются
`review`	Готов к рецензированию; обязательные body-разделы (§8.4.1) и type-specific (§8.5) присутствуют

`approved`	Архитектор подтвердил; `depends-on[]` консистенция проверена
`verified`	Все обязательные spec-specific TC (глава 9 §9.7) зелёные
`obsolete`	Заменён или больше не актуален; `replaced-by` обязателен

Связь с QG-0 / QG-2 SENAR:

- QG-0 (есть goal/AC у задачи) расширяется: для задач реализующих SPEC обязательны `implements-spec[]` в TR frontmatter.
- QG-2 (есть evidence у `done`) расширяется: для задач реализующих SPEC обязательны TC соответствующего spec-specific вида (глава 9 §9.7).

8.9 Схема хранения

8.9.1 На уровне системы

```
[requirements-substrate]/      # корень носителя требований (layout — guide/03
или guide/04)
  br/
  sr/
  specs/
    arch/   SPEC-ARCH-NN-*.md
    api/    SPEC-API-NN-*.md
    data/   SPEC-DATA-NN-*.md
    ui/     SPEC-UI-NN-*.md
    ai/     SPEC-AI-NN-*.md
    int/    SPEC-INT-NN-*.md
    proc/   SPEC-PROC-NN-*.md
    sec/    SPEC-SEC-NN-*.md
    ops/    SPEC-OPS-NN-*.md
  adapt/
  tz/
  SPECS.md                # auto-generated index
```

> *Все 9 типов SPEC (§8.3) допустимы на любом `level`; подпапки `specs/<type>/` создаются по мере необходимости — не все обязательны на уровне системы.*

8.9.2 На уровне подсистемы

```
[subsystem-substrate]/         # scope подсистемы
  br/                     # если своя бизнес-сторона
  sr/
  specs/
    arch/   SPEC-ARCH-NN-*.md      # архитектура подсистемы
    api/    SPEC-API-NN-*.md
    data/   SPEC-DATA-NN-*.md
```

```
    ui/     SPEC-UI-NN-*.md
    ai/     SPEC-AI-NN-*.md
    int/    SPEC-INT-NN-*.md
    proc/   SPEC-PROC-NN-*.md
    sec/    SPEC-SEC-NN-*.md
    ops/    SPEC-OPS-NN-*.md
  adapt/
  SPECS.md
```

Нативная для носителя реализация хранения специфична для носителя (см. guide/03, guide/04).

8.9.3 SPECS.md — auto-generated index

`SPECS.md` — auto-generated реестр всех SPEC: ID, тип, заголовок, статус, ссылка на верифицируемое требование, ссылка на файл. Помечается `linguist-generated=true`. Триггеры перегенерации — каждое изменение SPEC frontmatter или каждый approve / verify gate.

8.10 Связь с другими главами

Глава	Связь
02 Позиционирование методологии	SPEC как параллельная ось — следствие инверсии источника истины
06 Иерархия требований	`SR.constrained-by[]`, `TR.implements-spec[]`
07 ADAPT	SPEC ссылается на ADAPT через `source.adapt`
09 Тест-кейсы	Spec-specific TC types (таблица обязательных видов TC для каждого типа SPEC)
10 Жизненный цикл и QG	SPEC машина состояний + QG расширения для SPEC
03 Версионирование носителя	SPEC ID неизменяемы (V1); migration атомарно (V2)
11 Модель зрелости	RENAR-3+: все 9 типов SPEC где применимо
reference/02 — schemas	Полная machine-readable schema для каждого type-specific extension
reference/05 — knowledge graph schema	`constrained-by[]`, `implements-spec[]`, `depends-on[]` как edge types в графе

09. Тест-кейсы

> ***Часть RENAR Standard v1.0-draft*** · *← Оглавление*

9.1 Тест-кейс — полноценный артефакт

Тест в RENAR — не приписка в конце кода, а полноценный документ: у него своя версия, статус и место в цепочке прослеживаемости, как у требования, которое он проверяет. Причина проста: тесты пишет AI-агент, а AI охотно покрывает «счастливый путь» («ввели верный пароль — пустило») и тихо обходит неприятное («ввели чужой — не должно пустить, не должно подсказать, какое поле неверно, не должно записать пароль в лог»). Именно в необработанных негативных случаях и живут дефекты.

Поэтому RENAR делает нормативными два требования. **Парность pos/neg**: на каждое проверяемое утверждение — минимум один позитивный тест-кейс и один негативный (что должно произойти и что произойти не должно). **Изоляция судьи**: если результат оценивает другая AI-модель (для типов `ux`, `eval`), она обязана отличаться от той, что породила оцениваемое, — модель не проверяет сама себя. TC (Test Case) замыкает цепочку прослеживаемости T3 → ADAPT → BR / SR / SPEC → TR → TC (см. §2.3): от провала теста можно дойти до раздела T3, который он в конечном счёте проверяет.

Глава опирается на ISO/IEC/IEEE 29119 «Software testing» в части концепций test design, test execution, test result reporting и pos/neg coverage, но фиксирует закрытый список типов TC, обязательную pos/neg-парность и judge ≠ production isolation как нормативные требования v1.0, не присутствующие в ISO 29119 в формализованном виде.

От практик Specification by Example (Adzic) и BDD / Gherkin — где исполняемые примеры также служат спецификацией — RENAR отличается тем, что переводит pos/neg-парность (§9.7), judge ≠ production isolation (§9.13) и version-pin TC к версии требования (V5, §3.3.5) из рекомендации в блокирующие нормативные положения (§14.5.2).

Положения главы являются нормативными. Закрытые списки (принципы, типы TC, обязательные виды TC по типу SPEC) — обязательные положения соответствия RENAR (глава 13); расширение — только через формальную процедуру изменения стандарта.

> ***Плотная глава:*** *reference/09 · decision tree ниже — informative routing.*

9.1.1 Дерево решений: выбор `tc-type` (informative)

```
flowchart TD
  A[Новый TC для SR/SPEC] --> B{SPEC type?}
  B -->|SPEC-UI| C[tc-type: ux]
  B -->|SPEC-AI| D[tc-type: eval + adversarial negative]
  B -->|SPEC-API / INT / DATA| E[tc-type: contract]
  B -->|SPEC-SEC| F[tc-type: security — negative only]
  B -->|SPEC-ARCH / PROC / OPS| G[tc-type: system]
  B -->|BR acceptance| H[tc-type: acceptance]
  C --> I{pos/neg pair?}
  D --> I
```

```
  E --> I
  F --> J[security: negative mandatory; positive via system TC]
  G --> I
  H --> I
  I -->|SR normative claim| K[§9.7: pos + neg required]
  I -->|negative invariant only| L[§9.7 exception: single TC OK]
```

Процедура состязательного обзора TC — guide/07 §4.5; панель агентов (informative) — там же §4.5.

9.2 Закрытый список нормативных принципов TC

#	Принцип	Нормативная формулировка
P1	Полноценный артефакт (TC)	TC — самостоятельный артефакт стандарта, по жизненному циклу и версионированию равный требованию. Хранится отдельным файлом в подпапке `tests/` носителя требований (§9.17).
P2	Документ ≠ реализация	TC описывает **что и как проверяется** (отвязан от реализации). Реализация (код) адресуется полем `automation.location` и хранится в носителе кода. Один TC — одна реализация.
P3	AI-generated	TC создаются и редактируются AI-агентом по заданию инженера; инженер не пишет TC вручную (см. глава 11 §11.N).
P4	AI-executed	Все TC в статусе `ready` и выше — автоматизированы. Прогон выполняет автоматический runner (CI / AI-runner / specialized executor). Результаты в `last-run` записывает только runner (bot-managed участник) по факту прогона.
P5	Pos/neg pairing	На каждое утверждение требования (BR / SR / SPEC) создаётся минимум одна пара positive + negative TC (§9.7).
P6	`last-run` — bot-managed	Поле `last-run` (date / result / runner-id / requirement-version / judge-report) заполняет только автоматический runner. Ручное редактирование `last-run` любым участником запрещено стандартом (§9.12).
P7	Judge ≠ production isolation	Для типов TC, использующих LLM-as-judge (ux, eval), judge-модель **не должна совпадать** с production-моделью, генерирующей оцениваемый артефакт. Совпадение блокирует hook носителя (§9.6.2).

Список закрыт. Новые принципы добавляются только через формальную процедуру изменения стандарта (глава 13).

9.3 Общая схема TC (frontmatter)

Все типы TC делят общий набор frontmatter-полей. Type-specific поля добавляются как extensions поверх (§9.6). Полная machine-readable схема — в reference/02-schemas.md.

```
---
# === Identity (mandatory) ===
```

```yaml
id: TC-NN                           # immutable; NN sequential в рамках scope
title: "<short, descriptive>"
type: TC
slug: "<kebab-case>"                # auto-derived

# === Classification (mandatory) ===
tc-type: acceptance | ux | system | contract | eval | security
negative: boolean                   # true для парного негативного TC

# === Scope (mandatory) ===
level: system | subsystem | module
scope:
  system: "<system-id>"
  subsystem: "<subsystem-id>"       # null если level=system
  module: "<module-id>"             # null если level ≠ module

# === Lifecycle (mandatory) ===
status: draft | ready | passing | failing | obsolete

# === Verification target (mandatory; хотя бы один из) ===
verifies:
  - id: SR-NN | BR-NN | SPEC-<TYPE>-NN
    requirement-version: "<нативный для носителя version-ref>"   # V5 pinning
(см. глава 3)
  - id: ...

# === Pair link (mandatory если negative=false и существует парный) ===
paired-with:                        # ID парного TC (positive ↔ negative)
  - TC-NN

# === Automation (mandatory) ===
automation:
  status: automated | manual-pending
  location: "<нативный для носителя pointer to implementation>"  # mandatory если
automated
  manual-pending-until: "<ISO date>"                        # mandatory если
manual-pending
  manual-pending-reason: "<text>"                           # mandatory если
manual-pending

# === Execution (mandatory если type=ux | eval) ===
judge:
  vendor: "<provider>"              # mandatory; см. P7 isolation
  model: "<model-id>"
  prompt-template: "<template-path>@<version>"

baseline:                           # mandatory для ux | eval
  artifact: "<нативный для носителя pointer>"
  perceptual-diff-threshold: float  # для ux
  metric-thresholds: {}             # для eval

# === Last run (auto-managed; bot-only) ===
last-run:
  date: "<ISO-datetime>"
```

```
  result: pass | fail | skipped | n/a
  runner-id: "<runner-name@version>"
  run-ref: "<нативная для носителя ссылка>"
  requirement-version: "<version-ref of verified artifact>"
  judge-report: "<inline or pointer>"

# === AI provenance (обязательно на RENAR-4+; canonical schema — §4.10.1) ===
ai-provenance:
  generated-by: "<vendor>-<model>@<date>"
  generated-at: "<ISO-8601>"
  prompt-template: "<template-path>@<version>"
  context-tokens: integer
  output-tokens: integer
  human-edits: boolean
  # optional на RENAR-4, обязательно на RENAR-5 (см. §4.10.1):
  # cost-budget, cost-actual, generation-time-ms

# === Замена / obsolescence ===
obsolete-pending: boolean            # true при detected delta-T3 инвалидации
replaces: "<old-id>"
replaced-by: "<new-id>"
obsoleted-date: "<ISO date>"
---
```

`verifies[]` — закрытый список ссылок на верифицируемые артефакты (BR / SR / SPEC). TR прямо не указывается в `verifies` — TR верифицируется через свой родительский SR (см. §6.7). `verifies[].requirement-version` — нативный для носителя pinning артефакта (V5 capability, см. глава 3 §3.3.5); QG-2 (§9.10) требует совпадения `verifies[].requirement-version` с текущей версией артефакта.

9.4 Body разделы TC

Тело любого TC обязательно содержит следующие разделы (независимо от носителя):

Раздел	Обязательность	Содержание
Контекст	обязательно	На какой пункт верифицируемого артефакта ссылается TC; цитата или пересказ утверждения.
Предусловия	обязательно	Состояние системы и данных, требуемое для прогона; обеспечивается seed-механизмом.
Шаги	обязательно	Действия runner; для `tc-type: ux` — намерения, не селекторы (см. §9.6.1).
Pass-критерий	обязательно	Бинарный, наблюдаемый, воспроизводимый (см. §9.11).
Fail-критерий	обязательно	Перечень наблюдаемых признаков нарушения (не отрицание Pass); включает утечки, side-effects, race conditions.

Постусловия	обязательно	Какое состояние ожидается после прогона; cleanup-механизм.
Out of scope	обязательно	Что **намеренно** не проверяется, с указанием парного TC, где это покрыто.
Связанные TC	optional	Ссылки на семантически связанные TC.

Раздел «Out of scope» — нормативно обязательный: защищает от ложного ощущения покрытия. Отсутствие раздела блокирует переход TC в `ready`.

Имена секций тела — machine-detectable заголовки уровня `##`. Канонические идентификаторы секций критериев — `## Pass-критерий` и `## Fail-критерий`; именно их детектирует хук контроля change-of-criteria (§10.11.3), поэтому имена этих секций фиксированы и не подлежат локальной замене.

9.5 Закрытый список типов TC

```
tc-type ∈ { acceptance, ux, system, contract, eval, security }
```

Тип	Что проверяет	Применяется к	runner-семейство
`acceptance`	Достигнута ли бизнес-цель?	BR	E2E + AI-валидатор
`ux`	Соответствует ли UX заявленному опыту?	SPEC-UI	AI-driver + VLM-judge
`system`	Ведёт ли система себя как описано?	SR, SPEC-PROC, SPEC-ARCH	xUnit-семейство
`contract`	Соблюдён ли контракт?	SPEC-API, SPEC-INT, SPEC-DATA	Contract-testing framework
`eval`	Достигнуто ли качество AI-компонент?	SPEC-AI	Eval-runner с эталонным набором данных
`security`	Соблюдены ли security-инварианты?	SPEC-SEC	Authz/threat-test framework

Список закрыт. Новые типы добавляются только через формальную процедуру изменения стандарта (глава 13). Конкретные технологии runner специфичны для носителя и фиксируются в манифесте соответствия реализации.

9.6 Type-specific extensions

9.6.1 `tc-type: ux` — UX тесты на основе SPEC-UI

UX-тест нормативно построен как двухслойная структура:

Слой	Содержание	Исполнитель
Сценарий (намерение)	«<участник> хочет <результат> после <условие>»	AI-driver: переводит намерение в действия, находит элементы по семантике (без жёстких селекторов)
Перцептивная проверка	«На отрисованном состоянии видно <критерий>»	Perceptual judge (VLM): принимает рендер + критерий, возвращает pass/fail с обоснованием

Обязательное расширение frontmatter: `judge.vendor`, `judge.model`, `baseline.artifact`, `baseline.perceptual-diff-threshold`.

Обязательные секции тела дополнительно к §9.4: Сценарий (намерение, не селекторы); Перцептивный критерий (что должен увидеть judge); Парный негативный (пустое состояние / ошибка / отсутствие прав).

Регрессия по визуалу. Дополнительно к VLM-judge — perceptual diff против `baseline.artifact` с порогом `perceptual-diff-threshold`. Превышение порога блокирует переход TC в `passing`.

Обновление эталона. Изменение `baseline.artifact` требует нативного для носителя approval-механизма с тегом `[baseline-update]` (см. §9.13); автоматическое обновление запрещено.

9.6.2 `tc-type: eval` — Eval-тесты на основе SPEC-AI

Eval-тест нормативно проверяет качество AI-компонент через dataset с метриками и порогами.

Обязательное расширение frontmatter: `judge.vendor`, `judge.model`, `baseline.artifact` (versionable dataset), `baseline.metric-thresholds`.

Обязательные секции тела: происхождение dataset (как собран, какая разметка); метрика-кластер (одна eval-TC = одна семантически связная группа метрик; разные семейства — разные TC); regression rule (что считается провалом — выход за threshold или регрессия ≥ N% против эталона).

Judge ≠ production isolation (нормативно). Поле `judge.vendor` + `judge.model` обязано отличаться от `production-model` спецификации SPEC-AI, нормирующей оцениваемое поведение. Нативный для носителя hook (глава 3 §3.3.3) обязан блокировать merge change unit при совпадении.

Версионирование dataset. Eval-dataset — управляемый носителем versionable artifact: каждое изменение фиксируется как atomic change unit с описанием (что добавлено / убрано / переразмечено) и авторством (генератор-агент / критик-агент / human выборочная проверка).

Cost gating. Eval не запускается на каждое изменение реализации (cost): runner запускается при изменении SPEC-AI, production-model, или dataset, либо по расписанию. Триггеры фиксируются в нативной для носителя runner-configuration.

Двухступенчатая разметка dataset. Generator-agent создаёт кандидатов; critic-agent проверяет по чек-листу; инженер делает выборочную проверку ≥ 10% случайных примеров до merge dataset.

9.6.3 `tc-type: contract` — Контрактные тесты на основе SPEC-API / SPEC-

INT / SPEC-DATA

Обязательные секции тела: machine-readable контракт (ссылка на OpenAPI / GraphQL SDL / Protobuf / JSON Schema из SPEC); сторона генератор / сторона потребитель; мок counterparty (для SPEC-INT — sandbox / реальная среда отдельным TC).

Обязательное расширение для SPEC-INT. Контрактные TC обязательно сочетаются с интеграционным TC (`tc-type: contract` , `level: subsystem | system`) против реальной или sandbox-counterparty — мокированного контракта недостаточно для верификации SPEC-INT.

9.6.4 `tc-type: security` — Security-тесты на основе SPEC-SEC

Обязательные секции тела: атрибуты модели угроз (STRIDE-категория или эквивалент); subject под тестом (authn / authz / data classification / secrets / audit / encryption); negative scenarios (попытка обхода, неавторизованный доступ, leakage); ожидаемое поведение системы при нарушении.

Security-TC нормативно содержит **только negative scenarios** (попытка обхода защиты). Positive «дать корректный доступ корректному участнику» покрывается `tc-type: system` с областью охвата SPEC-SEC.

9.7 Pos/neg pairing — нормативное требование

На каждое утверждение требования (BR / SR / SPEC), описывающее наблюдаемое поведение, создаётся минимум одна пара positive + negative TC.

Положительный TC	Парный отрицательный TC
`negative: false`	`negative: true`
Описывает happy path / success behavior	Описывает граничные условия, нарушения, обходы
`paired-with: [TC-<neg-id>]`	`paired-with: [TC-<pos-id>]`

Negative TC нормативно описывает наблюдаемые признаки нарушения (что **не должно** произойти), а не отрицание Pass-критерия позитивного TC. Примеры:

Утверждение	Pos TC	Neg TC
«Аутентификация по email + password»	Корректные credentials → 200 + JWT	Неверный пароль → 401 без раскрытия, какое поле неверно; отсутствие записи в session-store; rate-limit после N попыток
«Создание заказа»	Валидный payload → 201 + order-id	Невалидный price < 0 → 422 с явной ошибкой; отсутствие записи в БД; отсутствие side-effect (уведомление, начисление)

QG-2 (§9.10) обязан блокировать promote артефакта в `verified` , если хотя бы одно нормативное утверждение покрыто только положительным TC.

Single-TC-coverage допускается **только** в одном случае: артефакт описывает запрет / negative invariant сам по себе (например, security-TC по STRIDE-категории — он негативный по природе).

9.8 Spec-specific TC — обязательные виды по типу SPEC

Закрытая нормативная таблица: каждый тип SPEC обязан иметь минимум по одному TC каждого «обязательного вида» перед переходом в `verified` (глава 8 §8.8).

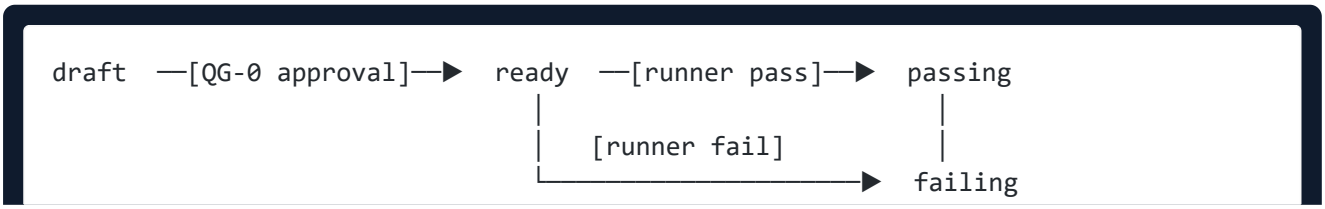
Тип SPEC	Обязательные виды TC	Дополнительные виды TC
SPEC-ARCH	Соответствие (zoning / dependency rules)	Эталонные значения атрибутов качества (latency / throughput / availability)
SPEC-API	Contract (по контракту из `contract-file`)	Auth negative; rate-limit; versioning compatibility
SPEC-DATA	Constraint (FK / NOT NULL / unique); Migration (прямой проход + откат)	PII handling; retention; index regression
SPEC-INT	Contract (mocked counterparty); контрактный TC `tc-type: contract` (real / sandbox counterparty)	Failure injection; idempotency; observability (correlation IDs)
SPEC-PROC	Happy path E2E; Alternative paths E2E	Compensation (для saga); SLA end-to-end
SPEC-UI	VLM-judge с эталоном (judge ≠ production); Accessibility (WCAG-AA минимум)	i18n (string overflow / RTL); Journey E2E
SPEC-AI	Eval против эталонного набора данных (judge isolated)	Состязательный (prompt injection как negative TC); Cost regression; Hallucination tests
SPEC-SEC	Authz / RBAC matrix; Threat-test per STRIDE-категории	Журнал аудита; Secrets leakage; Encryption invariants
SPEC-OPS	Smoke after deploy; SLO regression (load test)	Failover / DR drill; Observability (alert firing correctness)

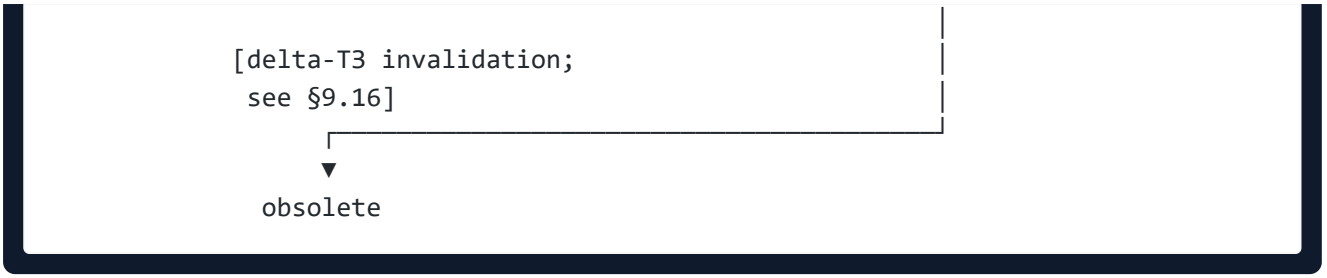
Нативный для носителя hook `promote SPEC → verified` (глава 3 §3.3.3) обязан проверять наличие минимум по одному TC каждого обязательного вида и блокировать переход при отсутствии.

Таблица закрыта на v1.0. Расширение — только через формальную процедуру изменения стандарта (глава 13).

9.9 Жизненный цикл TC

9.9.1 Машина состояний





Статус	Смысл	Триггер перехода
`draft`	Создан, реализация в работе	Создание AI-агентом
`ready`	Реализация валидна; dry-run runner прошёл; pos/neg парность подтверждена	QG-0 (§9.10): one-click approval
`passing`	Последний прогон `last-run.result = pass` на текущей `requirement-version`	Bot-managed по факту прогона
`failing`	Последний прогон `last-run.result = fail`	Bot-managed по факту прогона
`obsolete`	Терминальный; покрываемое поведение больше не существует	Delta-T3 инвалидация (§9.16) или deprecation родительского артефакта

`obsolete` — терминальный статус. TC в `obsolete` нативно для носителя сохраняется как исторический след: реализация носителя **должна** обеспечивать неизменность идентификатора TC и **не должна** допускать его переиспользование для нового TC (V1 capability; см. глава 3 §3.3.1).

9.9.2 Связь со статусом верифицируемого артефакта

Перевод BR / SR / SPEC в `verified` нормативно требует: все TC из `verified-by` верифицируемого артефакта имеют `last-run.result = pass` и `last-run.requirement-version` совпадает с текущей версией артефакта (см. §9.10 QG-2).

9.10 Контрольные точки качества для TC

Канонические определения Quality Gates — в глава 10 §10.3. Эта секция — TC-локальная сводка gates, применимых к TC напрямую (QG-0, QG-1) или использующих TC как доказательную базу (QG-2). Нумерация и семантика обязаны совпадать с канонической §10.3; локальное переопределение на уровне проекта запрещено (§10.10.2).

Gate (канонический)	Роль TC	Предусловие (кратко; полная формулировка — гл. 10)	Постусловие
QG-0 — утверждение (§10.3.1)	TC (`draft → ready`) — часть «утверждение»	Ссылка на `verifies[]` — артефакт есть в носителе в состоянии не ниже `approved` ; общие предусловия §10.3.1; решение утверждающего зафиксировано нативно для носителя (V3 + V6)	TC допускается к проверкам гейта реализации (QG-1)

QG-1 — реализация проверки (§10.3.2)	TC (`draft → ready`) — часть «реализация»	`automation.status = automated` (или `manual-pending` с дедлайном и причиной); pos/neg парность (§9.7); dry-run runner прошёл; обязательные секции body TC (§9.4) заполнены	TC переходит в `ready` ; допускается production-прогон runner
QG-2 — верификация (§10.3.3)	Артефакт (BR / SR / SPEC / TR) `→ verified` / `→ done` ; TC — доказательная база	Все TC из `verified-by` верифицируемого артефакта в статусе `passing` ; pos/neg парность по каждому нормативному утверждению; обязательные spec-specific виды TC (§9.8) присутствуют; `last-run.requirement-version` совпадает с текущей `version` артефакта	Верифицируемый артефакт переходит в `verified` (TR — в `done`); TC остаётся `passing`

Нативный для носителя one-click promote `draft → ready` атомарно проверяет предусловия обоих QG-0 и QG-1 как единый bundle (см. §10.3.2 «Триггер») — диаграмма §9.9.1 показывает агрегированное прохождение гейта как «утверждение QG-0».

Проверка совпадения `last-run.requirement-version` с текущей `version` верифицируемого артефакта при каждом последующем прогоне TC — runner-managed consistency check (§10.9.3), **не** отдельный Quality Gate в смысле §10.2.1. При несовпадении носитель автоматически переводит TC в `failing` до повторного прогона на актуальной версии артефакта; запись в audit-trail (§10.13) фиксируется типом `runner-fail` , не `gate-passage` .

Нативные для носителя hooks (глава 3 §3.3) обязаны блокировать gate-переходы, нарушающие предусловие.

9.11 Pass / Fail / Out of scope — нормативные критерии

9.11.1 Pass-критерий

Pass-критерий обязан быть:

- **Бинарным** — да или нет, без интерпретации.
- **Наблюдаемым** — фиксируется без доступа к внутренним структурам системы.
- **Воспроизводимым** — повторный прогон в тех же условиях даёт тот же результат.

Плохо	Хорошо
«Логин работает корректно»	« `POST /auth/login` с верными credentials возвращает 200 и JWT с `exp = now + 24h ± 1м` »
«Производительность приемлемая»	«p95 latency < 200 мс при 100 RPS на `/search` в течение 5 минут»
«Ошибка обрабатывается»	«При невалидном email возвращается 422 с телом `{"field":"email","code":"invalid_format"}` »

9.11.2 Fail-критерий

Fail-критерий — **не отрицание** Pass. Перечисляет наблюдаемые признаки нарушения, в том числе те, которые Pass-критерий явно не покрывает:

- Какой ответ / состояние / событие признаётся отказом.
- Утечки информации (например, в 401 не должно быть указано, какое именно поле credentials неверно).
- Side-effects, которых не должно быть — запись в лог, отправка письма, мутация других записей.
- Race conditions: одновременные запросы не приводят к нарушению инвариантов.

9.11.3 Out of scope

Каждый TC обязательно содержит секцию «Out of scope» с явным перечислением:

- Что намеренно не проверяется этим TC;
- Где это покрыто (ссылка на парный TC или другой TC).

Раздел защищает от ложного ощущения покрытия (см. также глава 11 — coverage matrix). Отсутствие явного «Out of scope» нормативно блокирует QG-0 (§9.10).

9.12 `last-run` — bot-managed only

Поле `last-run` (date / result / runner-id / run-ref / requirement-version / judge-report) заполняет **только** автоматический runner (CI-bot / AI-runner / specialized executor). Ручное редактирование `last-run` любым человеком — нарушение стандарта; hook носителя (глава 3 §3.3.6) обязан блокировать change unit, изменяющий `last-run` от автора, не являющегося ботом.

Состав `last-run`:

Поле	Обязательно	Содержание
`date`	да	ISO-datetime прогона
`result`	да	`pass | fail | skipped | n/a`
`runner-id`	да	Идентификатор runner + версия
`run-ref`	да	нативный для носителя pointer на полный лог прогона
`requirement-version`	да	Версия верифицируемого артефакта на момент прогона (V5 pinning)
`judge-report`	да для `ux | eval`	Inline или pointer на отчёт VLM/eval-judge

9.13 Защита от подгонки тестов

9.13.1 Нормативное правило

AI-агент не может одновременно изменить implementation-код и Pass / Fail-критерии существующего TC в одном change unit так, чтобы `failing` TC становился `passing`, без явного approval инженером на изменение TC.

Без этого правила AI-агент имеет тривиальный путь к зелёному прогону — ослабить критерий вместо исправления кода.

9.13.2 Механизм `[test-spec-change]`

Класс изменения	Тег	Утверждение
Изменение Pass / Fail-критериев существующего TC	`[test-spec-change]`	Обязательно: явный approval инженером change unit отдельно от любого изменения implementation-кода
Изменение `automation.location` (перенос реализации без изменения проверяемого поведения)	—	Без отдельного approval
Изменение implementation-кода без правок TC	—	Standard workflow
Обновление `baseline.artifact` / dataset / `mockup-baseline`	`[baseline-update]`	Обязательно: явный approval инженером
Создание нового TC	—	QG-0 (§9.10)

Нативная для носителя реализация тегов специфична для носителя (см. guide/03, guide/04); нормативное требование — атомарность change unit, явное обозначение класса изменения, и опциональная (но рекомендуемая) изоляция таких изменений в отдельный change unit от изменений implementation-кода.

9.13.3 Аудит

Все change units с тегом `[test-spec-change]` агрегируются в нативный для носителя audit-feed для архитектора. Цель — отследить паттерн: если AI-агент часто запрашивает изменение критериев, это сигнал о проблеме формулировок исходного требования (глава 7 ADAPT, категории обратных находок `terminology` или `gap`).

9.13.4 Judge isolation (P7) — частный случай защиты

`judge.vendor` + `judge.model` обязательно отличается от production-модели верифицируемого SPEC-AI (§9.6.2). Совпадение блокирует hook носителя.

9.14 Спот-чек инженера

9.14.1 Нормативная процедура

Раз в итерацию (по умолчанию — регулярный цикл реализации; конкретный интервал фиксируется в манифесте соответствия проекта) инженер выполняет спот-чек 5 случайных TC в статусе `passing`. Цель — поймать ситуацию, когда AI-агент сгенерировал «зелёный» TC, который ничего значимого не проверяет (`assert True`-эквивалент; VLM-prompt, проходящий на пустом экране; eval-критерий, всегда возвращающий pass на эталоне).

9.14.2 Sampling

Параметр	Нормативное требование
Размер выборки	5 TC (по умолчанию); может быть увеличено в манифесте соответствия проекта
Распределение по типам	Равномерное по `tc-type` (acceptance / ux / system / contract / eval / security) — каждый тип имеет шанс быть выбранным
Статус	Только `passing`
Кто выбирает	AI-агент случайным образом (нативная для носителя случайность; seed фиксируется в audit-feed)

9.14.3 Что инженер проверяет

1. Соответствует ли Pass / Fail-критерий заявленному поведению в верифицируемом артефакте?
2. Не слишком ли мягкий критерий VLM-judge или eval-judge?
3. Реальны ли предусловия (не подменены ли через seed, маскирующий bug)?
4. Покрывает ли Out-of-scope именно то, что должен покрывать парный TC?

9.14.4 Фиксация результата

Результат спот-чека фиксируется нативно для носителя:

```
last-выборочная проверка:
  date: "<ISO-date>"
  by: "<engineer-id>"
  sampled-tests: [TC-NN, TC-NN, ...]
  issues-found: integer
  issues:
    - test: "TC-NN"
      issue: "<краткое описание>"
```

9.14.5 Реакция на находки

При `issues-found > 0`:

- Архитектор регистрирует change unit на исправление найденных TC.
- AI-агент обязан учесть выявленный паттерн в следующих генерациях TC; паттерн добавляется в system prompt агента или в meta-style guide.
- При повторном обнаружении того же паттерна — эскалация на пересмотр шаблона генерации TC.

9.15 Матрица покрытия (auto-generated)

9.15.1 COVERAGE-artifact

`COVERAGE.md` (нативное для носителя имя artifact) — auto-generated сводный отчёт покрытия требований и спецификаций тест-кейсами на уровне носителя требований. Помечается нативным для носителя флагом auto-generated.

9.15.2 Обязательные метрики

Метрика	Цель	Действие при нарушении
`coverage-percent` (verified / total artifacts)	Целевой порог фиксируется в манифесте соответствия	нативный для носителя gate блокирует промоушн
`approved` без `verified`	0 перед промоушн	Бэклог AI-агента на следующую итерацию
Покрытие парным negative TC	100% утверждений	AI-агент создаёт change unit с парным негативом
`passing-tests / total-tests`	100% перед промоушн change unit	Блокирует QG-2 (§9.10)
`manual-pending` overdue	0	Уведомление архитектору; блокировка затронутых артефактов
Stale (`last-run.requirement-version` < текущей)	0	AI-агент перезапускает прогон

9.15.3 Триггеры перегенерации

`COVERAGE.md` перегенерируется автоматически при:

- Завершении change unit с изменением артефактов требований / SPEC / TC;
- Промоушн change unit в основную линию носителя;
- Каждом успешном прогоне runner (обновление `last-run`);
- По расписанию (нативный для носителя scheduler).

9.16 Delta-ТЗ и ТС

9.16.1 Impact analysis по тестам

При delta-ТЗ (глава 7 §7.6) AI-агент выполняет impact analysis по ТС одновременно с impact analysis по требованиям:

1. Находит все TC, у которых `verifies[].requirement-version` ниже новой версии верифицируемого артефакта.
2. Помечает их `obsolete-pending: true`.
3. Формирует таблицу затронутых TC в frontmatter delta-ADAPT или связанном change unit:

TC	Verifies	Старая версия	Новая версия	Действие
TC-NN	SR-NN	v1.1	v1.2	Update (новый шаг)
TC-NN	SR-NN	v1.1	v1.2	Без изменений (актуален)
TC-NN	BR-NN	v1.0	deprecated	Перевести в `obsolete`

4. Генерирует обновлённые версии TC в том же change unit, что delta-ADAPT.
5. После прогона обновлённых TC и их перехода в `passing` — снимает `obsolete-pending` и обновляет `verifies[].requirement-version` на новую версию артефакта.

9.16.2 Переход TC в `obsolete`

TC переходит в `obsolete` (терминально), если:

- Родительский артефакт (BR / SR / SPEC) переведён в `deprecated` без замены;
- Родительский артефакт заменён новым (`replaced-by`), для которого создан новый набор TC, **и** старый набор не покрывает поведения нового артефакта;
- Поведение, покрываемое TC, в новой версии артефакта больше не существует.

`obsolete` TC immutable и не удаляется (V1; см. глава 3 §3.3.1).

9.17 Схема хранения

Тест-кейсы хранятся в подпапке `tests/` носителя требований. Нативная для носителя реализация хранения специфична для носителя (см. guide/03 для distributed VCS; guide/04 для document-oriented store).

9.17.1 На уровне системы / подсистемы

```
[requirements-substrate]/      # system или subsystem scope (глава 6 §6.11)
  br/   sr/   tr/                # глава 6
  specs/                          # глава 8
  adapt/                          # глава 7
  tests/
    acceptance/  TC-NN-*.md
    system/      TC-NN-*.md
    ux/          TC-NN-*.md
    contract/    TC-NN-*.md
    eval/        TC-NN-*.md
```

```
    security/    TC-NN-*.md
    baselines/                    # для ux / eval
      <baseline-artifact>.png
      <eval-dataset>.jsonl
  COVERAGE.md                     # auto-generated (см. §9.15)
```

9.17.2 Реализация в субстрате кода

Реализация TC (код) живёт в носителе кода отдельно от носителя требований; адресуется полем `automation.location` (нативный для носителя pointer). Связь TC↔implementation: 1:1.

9.18 Связь с другими главами

Глава	Связь
02 Положение в типологии методологий	TC — нижний слой цепочки прослеживаемости (Утверждение 1 инверсия источника истины); pinning требования через `verifies[].requirement-version` (Утверждение 3 версионирование носителя)
06 Иерархия требований	TC верифицирует BR / SR; `verified-by[]` — auto-derived inverse edge на стороне требования
07 ADAPT	TC → SR → ADAPT → T3 — полная цепочка прослеживаемости; обратные находки (`terminology` , `gap`) питаются паттернами из `[test-spec-change]` audit (§9.13.3)
08 Specifications	Spec-specific TC по таблице §9.8; SPEC → `verified-by[]` auto-derived; type-specific extensions §9.6
10 Жизненный цикл и QG	машина состояний TC; QG-0 + QG-1 bundle для TC (`draft → ready`); QG-2 для верифицируемого артефакта требует pos/neg парность и spec-specific виды TC
03 Версионирование носителя	Immutable IDs (V1); atomic change unit и hooks (V2 + V3); diff & review для `[test-spec-change]` (V3); версионирование TC без потери истории (V4); pinning `verifies[].requirement-version` (V5); author + timestamp для `last-run` (V6)
11 Maturity model	RENAR-1: TC обязательно; RENAR-3+: pos/neg pairing 100%, spec-specific TC table обязательно; RENAR-4+: AI-generated + AI-executed
13 Соответствие	Закрытый список типов TC (§9.5) — обязательное положение v1.0; spec-specific TC table (§9.8) — обязательное положение v1.0; pos/neg pairing — обязательное положение v1.0; judge ≠ production isolation — обязательное положение v1.0
reference/02 — schemas	Полная machine-readable schema TC frontmatter + type-specific extensions

10. Жизненный цикл и Quality Gates

> ***Плотная глава:*** *читать после §6–§9; прохождение QG — guide/00; плотность — reference/09.*

10.1 Как артефакты движутся: статусы и гейты

Артефакт RENAR — требование, спецификация, ADAPT, тест — не лежит статично. Он движется по состояниям: `draft → approved → verified → …`. И двигаться как попало нельзя: каждый переход стережёт **контрольная точка качества** (Quality Gate) — условие, которое обязано выполниться, иначе переход не происходит. Требование не станет `verified`, пока все его тесты не позеленели на текущей версии; ADAPT не станет `approved` без двух подписей. Gate — это не «галочка успеха», а проверка, которая вправе сказать «нет» и оставить артефакт на месте.

Эта глава сводит воедино машины состояний всех артефактов и нормирует гейты: что проверяется перед каждым переходом, кто и когда обязан проверить. Глава фиксирует **только** состояния, переходы и гейты; frontmatter артефактов определяют главы 6–09.

10.1.1 Дерево решений: какой гейт сейчас (informative)

```
flowchart TD
  S[Изменение артефакта] --> T{Тип перехода?}
  T -->|draft → approved| Q0[QG-0 утверждение]
  T -->|approved → ready| Q1[QG-1 реализация проверки]
  T -->|ready → verified| Q2[QG-2 верификация]
  T -->|architecture sign-off| Q3[QG-3 опционально]
  T -->|client accept| Q4[QG-4 опционально]
  Q0 --> V0{предусловия §10.3.1}
  Q1 --> V1{TC automated §10.3.2}
  Q2 --> V2{passing-tests §10.3.3}
  V0 -->|fail| X[Остаётся draft]
  V1 -->|fail| X1[Остаётся approved]
  V2 -->|fail| X2[TC failing]
```

10.2 Нормативное определение Quality Gate

10.2.1 Quality Gate

Quality Gate (gate) — нормативное условие, проверка которого обязана быть выполнена для разрешённого перехода артефакта из одного состояния жизненного цикла в другое. Каждый gate состоит из:

1. **Идентификатор** — `QG-N` или `QG-<artifact>-<state>` (закрытый список §10.3, §10.4).

2. **Предусловие** — набор проверяемых утверждений об артефакте и связанных артефактах, которые обязаны быть истинны на момент запуска gate.
3. **Постусловие** — состояние, в которое переходит артефакт после успешного прохождения gate, и наблюдаемые эффекты (например, появление записи в лог переходов §10.13).
4. **Триггер** — кто или что инициирует проверку gate (участник: AI-агент / архитектор / автоматический runner; событие: одобрение / завершение прогона / поступление delta-T3).
5. **Точка контроля** — место в носителе, где проверка обязана быть автоматизирована (§10.11).

Gate не является событием успеха — это условие, которое **обязано быть проверено**. Прохождение gate может быть отрицательным (предусловие не выполнено) — в этом случае переход запрещён и артефакт остаётся в текущем состоянии.

10.2.2 Кто обязан проверять gate

Тип gate	Обязательный участник	Обеспечение соблюдения носителя
Утверждение (QG-0)	Архитектор или авторизованный носитель роли	Атомарная фиксация авторства и времени (V6, §3.3.6)
Реализация проверки (QG-1)	Автоматический runner (CI, eval-runner)	Атомарная фиксация результата прогона привязанного к версии артефакта (V5, §3.3.5)
Верификация (QG-2)	Автоматический runner с подтверждением `version-pin`	V5 + V6
Архитектура (QG-3, опционально)	Двойная подпись (клиент + архитектор)	V3 + V6
Приёмка (QG-4, опционально)	Заинтересованная сторона с полномочиями	V6

10.2.3 Связь с SENAR

SENAR §8 описывает Quality Gates как абстрактную концепцию для AI-управляемой разработки. RENAR **расширяет** SENAR в области инженерии требований:

- Сохраняет идентификаторы QG-0 / QG-1 / QG-2 как обязательные.
- Нормирует **формальные машины состояний** для каждого типа артефакта (SENAR этого не делает).
- Привязывает каждый переход в машине состояний к конкретному gate с предусловиями и постусловиями.
- Добавляет опциональные QG-3 / QG-4 для отраслей с расширенными требованиями к аудиту.

RENAR не противоречит SENAR; реализация SENAR-совместима с RENAR если выполнены требования §10.3 + §10.11.

10.3 Канонические RENAR gates (обязательные)

Закрытый список из трёх обязательных gates. Расширения вне этого списка — только опциональные §10.4 или через формальную процедуру изменения стандарта §10.10.

10.3.1 QG-0 — гейт утверждения

Назначение: разрешает переход артефакта из черновика в утверждённое для разработки состояние.

Предусловие (общая часть, дополняется per-artifact в §10.5–§10.9):

- Frontmatter артефакта валиден по схеме своей главы.
- Идентификатор артефакта уникален в носителе (V1, §3.3.1).
- Состязательный обзор произведён; или явно зафиксировано отсутствие применимости — допустимо **только** для тривиальных артефактов (по критериям, объявленным в манифесте соответствия, §13) с записью причины в лог переходов (§10.13).
- Если артефакт ссылается на источник (`source.adapt` для BR/SR/SPEC, `verifies[]` для TC) — ссылаемый артефакт существует в носителе в состоянии не ниже `approved` .

Постусловие:

- Артефакт переходит в `approved` (для requirements / SPEC) или `ready` (для TC) или `approved` ADAPT (§10.8).
- Запись в лог переходов (§10.13).
- Для requirements / SPEC: разрешается декомпозиция в дочерние артефакты (для BR — SR; для SR — TR + SPEC через `constrained-by` / `implements-spec`).

Триггер: явное одобрение архитектором / носителем роли в нативном для носителя механизме (V3 diff & review, §3.3.3).

Применимые артефакты: BR, SR, TR, SPEC, ADAPT, TC.

10.3.2 QG-1 — гейт реализации проверки (только TC)

Назначение: подтверждает, что для артефакта существует валидная реализация — код, конфигурация, инфраструктурный артефакт — пригодная для верификации.

Предусловие:

- Реализация привязана к версии артефакта через `version-pin` (V5, §3.3.5).
- `automation.status: automated` (с валидным `automation.location`) или `automation.status: manual-pending` (с указанным `manual-pending-until` и `manual-pending-reason`).
- Все статические проверки реализации агента носителя (типы, lint, схема) — пройдены.
- Pos/neg парность для покрываемых утверждений артефакта обеспечена (глава 9 §9.7).
- Все обязательные секции body TC (глава 9 §9.4) заполнены.

Постусловие:

- TC переходит в `ready` .
- Запись в лог переходов.

Триггер: одобряющий участник (one-click promote `draft → ready`) при подтверждении автоматического runner о прохождении dry-run.

Применимые артефакты: TC (`draft → ready`).

Примечание: TR не проходит отдельно гейт QG-1. Условия валидности реализации (impl scope, version-pin, статические проверки) для TR входят в предусловия QG-2 (§10.6.2). **QG-1 применим**

только к TC. Для BR / SR / SPEC переход `approved → verified` управляется единым QG-2; промежуточного гейта реализации QG-1 для требований и SPEC нет.

10.3.3 QG-2 — гейт верификации

Назначение: подтверждает, что наблюдаемое поведение системы соответствует артефакту: все TC из `verified-by` артефакта в состоянии `passing` на текущей версии артефакта.

Предусловие:

- Для BR / SR / SPEC: все TC из `verified-by` имеют `last-run.result = pass` и `last-run.requirement-version` (или эквивалент `spec-version` / `version`) совпадает с текущей `version` верифицируемого артефакта.
- Pos/neg парность по нормативным утверждениям артефакта — выполнена.
- Все обязательные spec-specific виды TC для типа артефакта присутствуют (глава 9 §9.8).
- Для TR: все его AC верифицированы привязанными TC (`last-run.result = pass`).
- Spec-specific дополнительные предусловия:
  - SPEC-UI / SPEC-AI: TC в состоянии `passing` с `judge-isolation` соблюдённой (глава 9 §9.13.4).
  - SPEC-SEC: TC `tc-type: security` присутствует и `passing`.

Постусловие:

- Артефакт переходит в `verified`.
- Запись в лог переходов с evidence-refs (список ID прогонов).
- Носитель обязан фиксировать `version` верифицируемого артефакта в evidence-записи (V5).

Триггер: автоматический runner подтверждает passing TC и инициирует promote-transition по запросу автора (one-click promote `approved → verified`).

Применимые артефакты: BR, SR, SPEC, TR.

10.4 Опциональные gates

QG-3 и QG-4 — нормативно описаны, но **не обязательны** для соответствия (глава 13). Реализация может объявить в манифесте соответствия либо поддержку QG-3 / QG-4, либо их отсутствие. Соответствие без QG-3 / QG-4 остаётся валидным.

10.4.1 QG-3 — гейт архитектуры (опционально)

Назначение: разрешает переход ADAPT из `answered` в `approved` (§10.8). Также применим к декомпозиционным решениям SPEC-ARCH в проектах с регулируемой архитектурной приёмкой.

Предусловие:

- Все обратные находки в ADAPT в статусе `resolved` (глава 7 §7.4.5).
- Двойная подпись готова: подпись клиента + подпись архитектора (глава 7 §7.5).
- Для SPEC-ARCH (если QG-3 применяется): декомпозиционное решение зафиксировано в носителе как ADR-like артефакт со ссылкой из SPEC-ARCH (форма ADR специфична для

носителя — fits в guide/).

Постусловие:

- ADAPT переходит в `approved` (immutable наравне с T3).
- Запись в лог переходов с обеими подписями (V6 author + timestamp фиксирует обоих участников).

Триггер: явное двойное утверждение; носитель обязан атомарно фиксировать обе подписи (V2 atomic change unit, §3.3.2).

Когда применять:

- ADAPT — всегда (но реализация может объявить QG-3 как локальный псевдоним для утверждения ADAPT, не выделяя его в отдельный gate).
- SPEC-ARCH — в проектах с регуляторными требованиями к архитектурной приёмке.

10.4.2 QG-4 — гейт приёмки (опционально)

Назначение: фиксирует приёмку клиентом бизнес-результата после релиза. Переход BR из `verified` в `accepted`.

Предусловие:

- BR в `verified` (QG-2 пройден).
- Измеримый бизнес-результат (`business-outcome` в frontmatter BR) — измерен; `current-value` зафиксирован.
- `achievement` ≥ project-configurable порога (по умолчанию 80%, фиксируется в манифесте соответствия).
- Формальная подпись заинтересованной стороны.

Постусловие:

- BR переходит в `accepted` (терминальный недеградируемый статус — обратный переход требует delta-T3).
- Запись в лог переходов с подписью заинтересованной стороны.

Триггер: формальная приёмка заинтересованной стороной по факту релиза.

Когда применять:

- Проекты с явной фиксацией post-release outcomes (продуктовые SaaS, регулируемые отрасли).
- При отсутствии QG-4 — статус `accepted` не используется; BR остаётся в `verified` до `deprecated`.

10.4.3 Соответствие стандарту с опциональными гейтами

Манифест соответствия (глава 13) обязан явно объявлять:

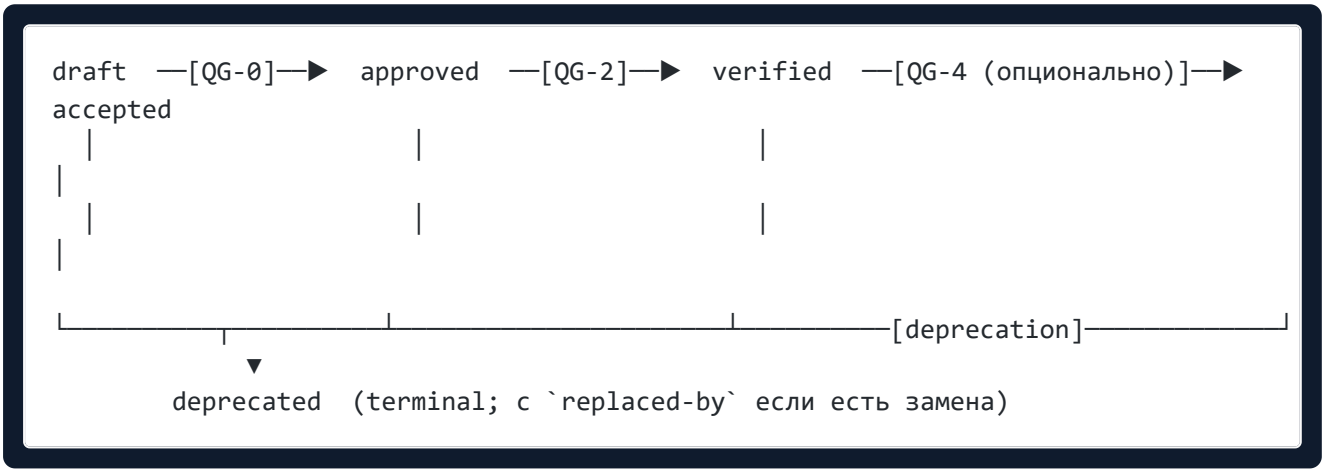
```
quality-gates:
  qg-0: required          # всегда required
  qg-1: required
  qg-2: required
```

```
  qg-3: declared          # required | declared | absent
  qg-4: declared
```

`declared` означает: реализация поддерживает gate; артефакты могут проходить его, но соответствие не требует прохождения для всех артефактов. `absent` — gate не применяется в реализации; терминальное состояние артефакта — `verified` (без `accepted`).

10.5 Машина состояний BR / SR

10.5.1 Состояния и переходы



Статус	Семантика	Gate перехода
`draft`	Создан AI-агентом или архитектором; ещё не утверждён	— (создание)
`approved`	Утверждён к декомпозиции / реализации	QG-0 (§10.3.1)
`verified`	Все производные TC `passing` на текущей версии	QG-2 (§10.3.3)
`accepted`	Post-release бизнес-результат подтверждён	QG-4 (§10.4.2, опционально)
`deprecated`	Терминальный; не удаляется (V1 immutable history)	Deprecation transition (§10.5.3)

Frontmatter BR / SR (включая обязательные поля статуса) определяется в главе 6 §6.5.2 / §6.6.2. Этот раздел нормирует **только** переходы между состояниями и привязку к gates.

10.5.2 Предусловия по переходам

Переход	Gate	Дополнительные предусловия (поверх §10.3)
`draft → approved`	QG-0	BR: `business-outcome` заполнен. SR: `parent` BR в состоянии не ниже `approved`. Если SR ссылается на SPEC через `constrained-by[]` — все SPEC в `approved` или выше.

`approved → verified`	QG-2	Хотя бы один TC с `negative: true` в `verified-by`. Все `last-run.requirement-version` совпадают с текущей `version`.
`verified → accepted`	QG-4	Только если реализация объявила QG-4.
`* → deprecated`	Deprecation transition (§10.5.3)	См. §10.5.3

10.5.3 Deprecation transition

Предусловие:

- Артефакт находится в любом не-`deprecated` состоянии.
- `replaced-by` (если указан) существует в носителе и находится в состоянии не ниже `approved`.
- Нет активных дочерних TR в состоянии `approved` или активных задач исполнителя по этому артефакту (атомарная переадресация задач на replacement — обязательное условие, V2 atomic change unit).

Постусловие:

- `status: deprecated`.
- `deprecated-date` записан (V6).
- `replaced-by` указан, если есть замена.

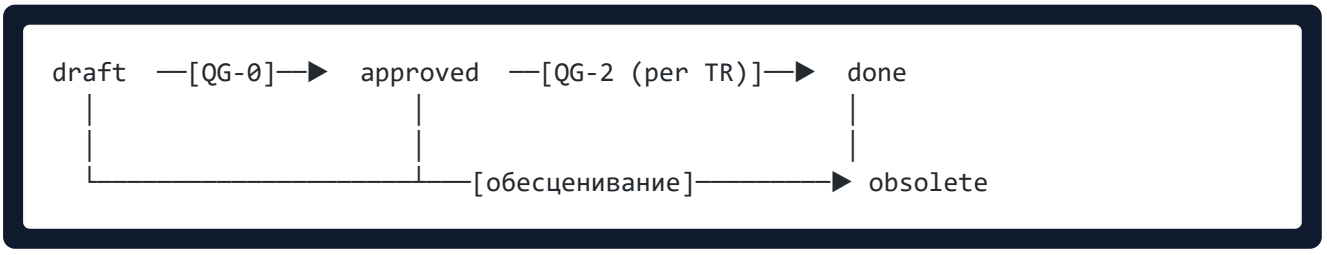
Триггер: архитектор или Владелец продукта.

10.5.4 Reverse-эволюция верификации

Если артефакт уже `verified`, но `version` инкрементировался (например, после применения delta-ADAPT, глава 7 §7.6) — статус обязан вернуться в `approved` до повторного прохождения QG-2 на новой версии. Этот переход обязателен и автоматический: носитель обязан инвалидировать `verified` при изменении `version` (глава 6 §6.5.4 reverse-эволюция).

10.6 Машина состояний TR

10.6.1 Состояния и переходы



Статус	Семантика	Gate

`draft`	TR создан; AC ещё не финализированы	—
`approved`	AC утверждены; разрешён старт работы исполнителя	QG-0 (§10.3.1) с предусловиями TR из §10.6.2
`done`	AC верифицированы; все привязанные TC `passing`	QG-2 (§10.3.3) с предусловиями TR из §10.6.2
`obsolete`	TR утратил актуальность до завершения (родительский SR изменился)	Обесценивание (архитектор)

10.6.2 Предусловия для TR

QG-0 для TR (`draft → approved`) — дополнительно к общей части:

- Цель сформулирована (`goal`).
- AC верифицируемы и независимы (каждый AC — отдельная проверка).
- Хотя бы один негативный сценарий присутствует.
- Установлена ссылка на parent SR (или BR для простых конфигураций) через `implements` .
- Если TR реализует SPEC — обязательное поле `implements-spec[]` (глава 8 §8.6.2).
- Если задача затрагивает безопасность — `threat-surface` задекларирован (глава 8 §8.5.8).

QG-2 для TR (`approved → done`):

- Все AC TR подтверждены прохождением соответствующих TC (`last-run.result = pass`).
- Pos/neg парность по каждому AC.
- Для TR реализующего SPEC: TC соответствующего spec-specific вида существует и `passing` (глава 9 §9.8).

10.6.3 Обесценивание TR

Если родительский SR переходит в `deprecated` или его `version` изменилась так, что AC TR более не актуальны — TR переходит в `obsolete` . Это **не** деградация — это альтернативный терминальный путь. TR в `obsolete` не удаляется.

10.7 Машина состояний SPEC

10.7.1 Состояния и переходы

```
draft  ──[review-transition]──▶  review  ──[QG-0]──▶  approved  ──[QG-2]──▶
verified
                                                          |
|
```

```
                                                    └──[обесценивание]──▶
obsolete  (terminal)
```

Статус	Семантика	Gate
`draft`	SPEC создан; обязательные frontmatter-поля заполняются	—
`review`	Обязательные body-разделы (§8.4.1) и type-specific (§8.5) присутствуют; готов к рецензированию	Review-transition (§10.7.2)
`approved`	Архитектор подтвердил; `depends-on[]` консистентен	QG-0
`verified`	Все обязательные spec-specific TC `passing`	QG-2
`obsolete`	Заменён или больше не актуален; `replaced-by` обязателен	Deprecation (§10.5.3 mutatis mutandis)

10.7.2 Review-transition (`draft → review`)

Review-transition не является полноценным gate в смысле §10.2.1 — это автоматическая проверка структурной полноты. **Предусловие**:

- Все обязательные frontmatter-поля §8.4 заполнены.
- Все обязательные body-разделы §8.4.1 присутствуют.
- Type-specific разделы §8.5 присутствуют для соответствующего `spec-type`.

Постусловие: `status: review`; артефакт виден архитектору для рецензирования.

Если review-transition не пройден — артефакт остаётся в `draft`; носитель обязан вернуть список отсутствующих секций (V3 diff & review поддерживает структурный фидбек).

10.7.3 Предусловия для SPEC

QG-0 для SPEC (`review → approved`) — дополнительно к общей части §10.3.1:

- `depends-on[]` граф ацикличен (глава 8 §8.6.3).
- Все SPEC из `depends-on[]` в состоянии не ниже `approved`.
- Если SPEC ссылается на ADAPT через `source.adapt` — ADAPT в состоянии `approved`.

QG-2 для SPEC (`approved → verified`):

- Все обязательные spec-specific виды TC для `spec-type` присутствуют и `passing` (глава 9 §9.8).
- Для SPEC-AI: pos/neg pair coverage по `evaluation-criteria` = 100%; judge-isolation соблюдена.
- Для SPEC-SEC: `tc-type: security` присутствует.
- Для SPEC-DATA: `tc-type: contract` присутствует для опубликованных interface-полей.

10.8 Машина состояний ADAPT

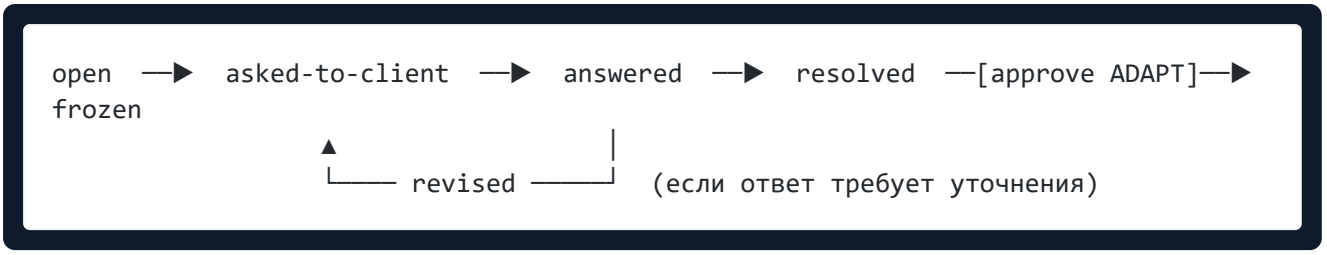
10.8.1 Macro-состояния

Состояния ADAPT (`draft → review → client-ready → answered → approved → frozen` , и терминальное `superseded` при дезавуировании) определены в главе 7 §7.4 и §7.6.4. Этот раздел нормирует gates.

Переход	Gate	Предусловие
`draft → review`	Review-transition	Прямая интерпретация охватывает все разделы T3; первичные обратные находки зафиксированы в `open`
`review → client-ready`	Backward-ready	Все обратные находки переведены в `asked-to-client` ; пакет вопросов сформирован
`client-ready → answered`	Client-return	Все обратные находки в `answered` с author + timestamp ответа клиента (V6)
`answered → approved`	**QG-3** (§10.4.1)	Все обратные находки в `resolved` ; двойная подпись готова
`approved → frozen`	Freeze-transition	Автоматический после approve; ADAPT immutable; разрешается генерация BR / SR / SPEC с `source.adapt = approved`
`frozen → superseded`	**QG-3 дезавуирующего ADAPT** (§10.8.5)	Дезавуирующий ADAPT (`supersedes: ADAPT-MMM`) достиг `approved` ; производные BR / SR / SPEC перенаправлены или пере-выведены

10.8.2 Вложенная машина состояний для записи об обратной находке

Каждая запись об обратной находке внутри ADAPT имеет собственную подчинённую машину состояний (глава 7 §7.4.5):



Sub-state	Семантика
`open`	Записан инженером; не отправлено клиенту
`asked-to-client`	Отправлен клиенту; зафиксирована дата вопроса
`answered`	Клиент ответил; ответ записан (V6 author + timestamp)
`resolved`	Инженер интегрировал ответ в прямую интерпретацию
`revised`	Ответ расплывчатый; повторный вопрос (возврат к `asked-to-client`)
`frozen`	После approval ADAPT; изменения невозможны

Нормативное правило: QG-3 (approve ADAPT) **запрещён**, если хотя бы одна запись об обратной находке в `open` / `asked-to-client` / `answered` / `revised`. Все такие записи обязаны быть в `resolved` (глава 7 §7.4.5).

10.8.3 QG-3 для ADAPT — детализация

Предусловие (полная):

- Все разделы прямой интерпретации заполнены (критерий forward complete).
- Все записи об обратных находках в `resolved`.
- Подпись клиента получена и зафиксирована нативным для носителя механизмом (V3 + V6).
- Подпись архитектора получена и зафиксирована.
- Если ADAPT — delta-ADAPT: parent-ADAPT в `frozen` (глава 7 §7.6).

Постусловие:

- ADAPT переходит в `approved`.
- Запись в лог переходов с обеими подписями.
- Носитель обязан атомарно (V2) зафиксировать обе подписи: частичная подпись (только клиент / только архитектор) **не** переводит ADAPT в `approved`.

Триггер: явное одобрение обоими участниками.

10.8.4 Errata для frozen ADAPT

`frozen` — терминальное состояние по линии деривации. Изменения возможны только добавлением нового артефакта по одному из трёх путей:

1. **Delta-ADAPT** (если T3 содержит ambiguity, обнаруженную поздно) — новый артефакт с явной связью `parent-adapt`.
2. **Errata-ADAPT** (если ошибка интерпретации инженера) — отдельный артефакт с подписью клиента (если меняется контрактный итог) или только архитектора (если косметическая).

3. **Дезавуирование** (если прежнее решение было верным, но позже опровергнуто) — дезавуирующий ADAPT переводит дезавуируемый в `superseded` (§10.8.5, глава 7 §7.6.4).

Во всех трёх случаях frozen ADAPT **не редактируется**. Это требование V1 (immutable history) для контрактных артефактов (глава 7 §7.6.3).

10.8.5 Переход frozen → superseded (дезавуирование)

Дезавуирование approved/frozen ADAPT нормировано в главе 7 §7.6.4. Переход жизненного цикла:

```
frozen ──[дезавуирующий ADAPT достиг approved через QG-3]──▶ superseded
(терминальное, неизменяемое)
```

Отдельный QG не вводится. Дезавуирование проходит через тот же **QG-3** (двойная подпись, §10.8.3), что и обычный ADAPT — дополнительной контрольной точки не создаётся.

Предусловие перехода `ADAPT-MMM → superseded`:

- Создан дезавуирующий `ADAPT-NNN` с полем `supersedes: ADAPT-MMM` и непустым `supersession-rationale` (глава 7 §7.6.4).
- Дезавуирующий ADAPT прошёл QG-3 и достиг `approved`. Если дезавуируемое решение имело контрактный итог (типовой случай) — двойная подпись **обязана** включать подпись клиента; подпись только архитектора допустима лишь для строго косметической правки без контрактного итога.
- Все производные `BR` / `SR` / `SPEC` с `source.adapt: ADAPT-MMM` перенаправлены на дезавуирующий ADAPT либо пере-выведены (нет висячих ссылок).

Постусловие:

- `ADAPT-MMM` переходит в терминальное **`superseded`** — отдельное от `obsolete` и от `frozen`; неизменяемо и **сохраняется** для аудита (V1), не удаляется.
- Поле `superseded-by: ADAPT-NNN` на `ADAPT-MMM` фиксируется автоматически.
- Запись в лог переходов с подписями дезавуирующего ADAPT (V6).

Триггер: утверждение дезавуирующего ADAPT через QG-3.

Hook-обязательство: после перехода висячая ссылка `source.adapt` на ADAPT в статусе `superseded` — **fatal**; обеспечение соблюдения — `adapt-supersession` validation (§10.11.1), гейт `check-adapt-supersession.js`.

10.9 Машина состояний TC

10.9.1 Состояния и переходы

```
draft  ──[QG-0]──▶  ready  ──[runner pass]──▶  passing
                      │                          │
                      │  [runner fail]           │ [criteria change |
```

Статус	Семантика	Gate / триггер
`draft`	TC создан; реализация в работе	—
`ready`	dry-run runner прошёл; pos/neg парность подтверждена	QG-0 (§10.3.1) с предусловиями TC из §10.9.2
`passing`	`last-run.result = pass` на текущей `requirement-version`	runner pass; bot-managed
`failing`	`last-run.result = fail`	runner fail; bot-managed
`obsolete`	Покрываемое поведение более не существует	Deprecation (§10.9.4)

Frontmatter TC и pos/neg pairing определяются в главе 9.

10.9.2 Предусловия для TC

QG-0 для TC (`draft → ready`) — дополнительно к общей части:

- `automation.status: automated` (с валидным `automation.location`) ИЛИ `automation.status: manual-pending` (с `manual-pending-until` ≤ +1 sprint и заполненным `manual-pending-reason`).
- Pos/neg парность по покрываемым утверждениям подтверждена (§9.7).
- Dry-run runner прошёл (только структурная валидность; не путать с production-прогоном).
- Все обязательные секции body TC (§9.4) заполнены.
- Citation на `verifies[]` — артефакт существует в носителе в состоянии не ниже `approved` .

Постусловие:

- `status: ready` .
- Runner разрешён к production-прогону.

10.9.3 Переходы под управлением runner (не Quality Gates)

`ready → passing` , `ready → failing` , `passing → failing` , `failing → passing` — переходы, происходящие **только** по факту прогона runner (глава 9 §9.12 `last-run` bot-managed). Эти переходы **не являются** Quality Gates в смысле §10.2.1: они — нормативные следствия результатов прогона, а не прохождение гейта с предусловиями и постусловиями. В частности, проверка совпадения `last-run.requirement-version` с текущей версией верифицируемого артефакта (см. §9.10) — это проверка согласованности под управлением runner, а не отдельный gate.

Постусловие каждого runner-перехода:

- `last-run` обновлён: `result`, `timestamp`, `requirement-version`, `evidence-refs`.
- Носитель обязан запретить ручную модификацию `last-run` (только runner-actor).

10.9.4 Обесценивание TC

TC переходит в `obsolete` если:

1. Артефакт из `verifies[]` переходит в `deprecated` / `obsolete`.
2. Delta-T3 инвалидирует тестовое поведение (глава 9 §9.16).

Постусловие: `status: obsolete`. TC не удаляется (V1 immutable history).

10.9.5 Change-of-criteria — отдельный нормативный путь

Изменение `## Pass-критерий` или `## Fail-критерий` в TC — **не обычный transition**; это специальный путь, требующий отдельного approval workflow (глава 9 §9.13). Подробности обеспечения соблюдения — §10.11.3.

10.10 Политика закрытого списка

10.10.1 Нормативное правило

Закрытый список Quality Gates RENAR — обязательные {QG-0, QG-1, QG-2} и опциональные {QG-3, QG-4}. Изменение списка возможно **только** через формальную процедуру изменения стандарта RENAR (глава 13).

Настоящая политика является специализацией §1.7 Политика закрытого списка для квалити-гейтов; общее правило для всех закрытых списков RENAR и master-индекс — §1.7.5.

10.10.2 Что запрещено

Действие	Запрещено?	Почему
Локальное создание нового gate-типа `QG-N` на уровне проекта	Запрещено	Нарушает закрытый список; делает соответствие не-портируемым
Локальное переопределение предусловий канонического гейта	Запрещено	Делает соответствие несравнимым между реализациями
Дополнительное ужесточение предусловий локального гейта	Разрешено	Манифест соответствия может объявить более строгие пороги (например, `qg-2.required-negative-tc: true`)
Локальное ослабление предусловий канонического гейта	Запрещено	Нарушает контракт стандарта
Объявление QG-3 / QG-4 как `absent` в манифесте соответствия	Разрешено	Опциональные gates — §10.4

Объявление QG-0 / QG-1 / QG-2 как `absent`	Запрещено	Нарушает соответствие §10.4.3

10.10.3 Расширение списка

Добавление нового gate-типа возможно через:

1. Запрос на изменение стандарта с обоснованием — исследовательский черновик с типологией и сравнением с каноническими gates.
2. Публичное обсуждение (срок и форум фиксируется политикой стандарта, глава 13).
3. Включение в следующую minor-версию стандарта (`v1.X` или `v2.0`).

Локальные для проекта расширения остаются за пределами соответствия — они допустимы как internal-практики, но **не** влияют на манифест соответствия.

10.11 Независимое от носителя обеспечение соблюдения

10.11.1 Нормативные требования

Носитель, реализующий RENAR, обязан обеспечить автоматическую проверку предусловий гейтов на следующих точках:

Точка контроля	Что обязано быть проверено	Опирается на capabilities
Promote-transition (любой переход в более высокий статус)	Предусловия соответствующего gate (§10.3, §10.4, §10.5–§10.9)	V3 (diff & review) для блокировки перехода до approve; V4 (branching) для отделения WIP от утверждённой правды
Approve-transition (любое действие утверждения)	Авторство зафиксировано (actor) и временная метка	V6 (author + timestamp)
Reference-validation (любое создание/ изменение артефакта с ссылкой на другой)	Referenced артефакт существует и в требуемом состоянии	V1 (immutable history) для stable identifier; V5 (version pin) для межносительных ссылок
Change-of-criteria для TC (§10.11.3)	Применён отдельный процесс утверждения	V3 + V6
Runner-transitions для TC (`ready → passing` / `failing`)	Только runner-actor может писать `last-run`	V6 (authorship); нативный для носителя ACL или role-based restrictions
Инвалидация жизненного цикла	Артефакт автоматически переведён в `approved`	V5 (version pin) для детекции

(артефакт `verified`, версия инкрементировалась)		
`implements`-edge validation (BR подсистемы со ссылкой на BR системы, §6.5.2, §6.8.2)	(1) target BR существует по `id + scope.system`; (2) target в `approved` + при approve данного BR (`deprecated` target — warning, не fatal; cascade-warning по `implemented-by[]`); (3) цепочка `implements` не образует циклов; (4) `implements[]` не присутствует при `level: system`	V1 (stable identifier для target lookup); V3 (block approve до пройденной валидации); V5 (cross-substrate ID разрешение когда target в другом носителе)
`adapt-applicability` validation (§7.4.1)	(1) Для каждого ТЗ зафиксирован вердикт состязательного обзора (V6 author + timestamp). (2) Если вердикт «findings present» — должен существовать соответствующий ADAPT в `approved` + с двойной подписью; BR/SR/SPEC производные имеют `source.adapt`. (3) Если вердикт «no findings, no clarifications» — ADAPT отсутствует, BR/SR/SPEC имеют `source.tz-section` + `source.adversarial-review-ref` evidence. (4) Запрет смешения: artifacts с `source.adapt` omitted без свидетельства вердикта — fatal.	V3 (block approve до прохождения validation); V6 (вердикт + signature attribution); V1 (вердикт immutable evidence)
`adapt-supersession` validation (§7.6.4, §10.8.5)	(1) `supersedes: ADAPT-MMM` ссылается на существующий ADAPT; обратная связь `superseded-by` симметрична. (2) `supersession-rationale` непустой и ссылается на конкретное противоречащее `BR` / `SR` / `SPEC`. (3) При контрактном итоге дезавуируемого решения — дезавуирующий ADAPT имеет подпись клиента. (4) Висячая ссылка `source.adapt` на ADAPT в статусе `superseded` — **fatal**.	V1 (stable identifier + immutable superseded history); V3 (block approve до пройденной валидации); V6 (signature attribution)

10.11.2 Носитель без V3 / V4 / V6 — не соответствующий

Носитель, не обеспечивающий V3 (diff & review), не может реализовать gates: нет способа отделить «предложенное изменение» от «утверждённой правды» (глава 3 §3.3.3). Аналогично V4 (branching, §3.3.4) и V6 (author + timestamp, §3.3.6) — без них механика утверждения невозможна. Носитель, не удовлетворяющий V3 / V4 / V6, **не реализует RENAR** независимо от других свойств.

10.11.3 Change-of-criteria для TC — особое обеспечение соблюдения

Изменение Pass / Fail критерия TC — высокорисковая операция (защита от test-fitting, глава 9 §9.13). Носитель обязан:

1. **Детектировать**: любое изменение секций `## Pass-критерий` / `## Fail-критерий` в TC-артефакте.
2. **Принудительно изолировать**: change-of-criteria обязано быть отдельным change-set (V4 atomic change unit), помеченным признаком, отличающим его от обычных правок (нативный

для носителя механизм — частный случай V3 diff & review; форма признака специфична для носителя, выносится в guide/).
3. **Запретить совмещение**: одно и то же лицо не имеет права одобрить change-of-criteria и одобрение fix кода, который тестируется этим же TC. Носитель обязан проверять это правило при approve-transition.
4. **Регистрировать**: change-of-criteria записывается в audit-trail (§10.13) с явной типизацией события.

10.11.4 Формы нативной для носителя реализации

Конкретные нативные для носителя механизмы (как именно реализуется hook в данном носителе) — выносятся в `guide/` и манифест соответствия. Стандарт не нормирует **форму** hook (это специфичное для носителя решение). Стандарт нормирует **что hook обязан проверять и в какой точке**.

Раздел `guide/` обязан содержать для каждого поддерживаемого носителя:

- Mapping точек обеспечения соблюдения §10.11.1 на нативные для носителя механизмы.
- Примеры реализаций.
- Известные ограничения носителя относительно автоматизации каждой проверки.

10.12 Запрещённые переходы

Закрытый список переходов, нарушающих жизненный цикл. Носитель обязан их блокировать.

Из	В	Артефакт	Почему запрещено
`draft`	`verified`	BR / SR / SPEC	Пропускает QG-0; нет evidence approval
`draft`	`accepted`	BR	Same; и пропускает QG-2
`draft`	`done`	TR	Пропускает QG-0
`draft`	`passing`	TC	Пропускает QG-0 (нет dry-run runner)
`obsolete`	*	Любой	Терминальный статус; «реанимация» запрещена — нужен новый артефакт с `supersedes`
`deprecated`	*	Любой	Same
`frozen`	*	ADAPT	Same; изменения только через delta-ADAPT или errata (§10.8.4)
`verified`	`draft`	BR / SR / SPEC	Деградация через несколько ступеней — потенциальная потеря trace; если требуется доработка — `verified → approved` через delta или reverse-эволюция §10.5.4
`accepted`	`verified`	BR	Деградация после приёмки — недопустима без delta-T3
`accepted`	`approved`	BR	Same

`passing`	`draft`	ТС	Деградация теряет history runs; используется `passing → failing → obsolete` или change-of-criteria путь
`ready`	`draft`	ТС	Деградация теряет dry-run evidence (§10.9.2); ослабление ТС через `[test-spec-change]` (глава 9 §9.13) — не путь возврата в `draft`
`failing`	`draft`	ТС	Деградация теряет runner history; повторная диагностика — через новый прогон (`failing → passing` runner-managed) или `obsolete`

10.12.1 Реакция носителя

При попытке запрещённого перехода носитель обязан:

1. Заблокировать transition (V3 diff & review).
2. Вернуть вызывающему участнику код ошибки с указанием конкретного нарушенного правила (по идентификатору строки этой таблицы).
3. Не создавать запись в лог переходов (§10.13).

10.13 Логирование событий прохождения gate

10.13.1 Нормативное требование

Каждое успешное прохождение gate (любого типа: QG-0, QG-1, QG-2, QG-3, QG-4, runner-transition, deprecation, freeze-transition) обязано быть зафиксировано в носителе как неизменяемое событие со следующими полями:

Поле	Семантика	Обязательность
`timestamp`	UTC ISO-8601 момент успешного прохождения	Обязательно
`artifact-id`	Идентификатор артефакта (immutable, V1)	Обязательно
`artifact-type`	`BR` / `SR` / `TR` / `SPEC-<type>` / `ADAPT` / `TC`	Обязательно
`artifact-version`	Версия артефакта на момент перехода (V5)	Обязательно
`from-status`	Исходное состояние	Обязательно
`to-status`	Целевое состояние	Обязательно
`gate-id`	`QG-0` / `QG-1` / `QG-2` / `QG-3` / `QG-4` / `deprecation` / `freeze` / `runner-pass` / `runner-fail` / `change-of-criteria`	Обязательно
`actor`	Идентификатор инициатора (V6); для двойной подписи — список участников	Обязательно

`evidence-refs`	Ссылки на доказательства: ID прогонов runner, ID артефактов состязательного обзора, ID подписей	Обязательно для QG-2 / QG-3 / QG-4
`notes`	Свободный текст	Опционально

10.13.2 Независимый от носителя формат

Формат хранения событий — специфичен для носителя (отдельный лог-стрим / append-only collection / иные формы). Стандарт нормирует только обязательные поля §10.13.1, не их сериализацию.

Манифест соответствия обязан указать механизм хранения событий и формат экспорта (для аудита, глава 13).

10.13.3 Retention

События **не удаляются** в течение всего жизненного цикла артефакта и после его перехода в `deprecated` / `obsolete` / `frozen`. Этого требует V1 (immutable history) и нормативные положения compliance (глава 13).

10.14 Связь с другими главами

Глава	Связь
02	SENAR QG-0..QG-2 — концептуальная основа; RENAR расширяет (§10.2.3)
06	frontmatter и body BR / SR / TR (§6.5–§6.7); state machines детализированы здесь (§10.5–§10.6)
07	frontmatter ADAPT (§7.8); backward sub-states (§7.4.5); двойная подпись (§7.5); delta-ADAPT (§7.6) — машина состояний здесь (§10.8)
08	frontmatter SPEC (§8.4); type-specific QG (§8.8); машина состояний здесь (§10.7)
09	frontmatter TC (§9.3); pos/neg pairing (§9.7); change-of-criteria (§9.13); машина состояний здесь (§10.9)
03	V1–V6 — фундамент обеспечения соблюдения (§10.11); без V3 / V4 / V6 — нет реализации gates
11	Уровни зрелости определяют объём применимых gates (например, RENAR-1 — QG-0 / QG-1 / QG-2 обязательны; на более высоких уровнях усиливаются предусловия QG-2: pos/neg парность, spec-specific TC)
13	Манифест соответствия объявляет gate support (§10.4.3); хранит retention policy событий (§10.13.3)

11. Модель зрелости

> ***Часть RENAR Standard v1.0-draft*** · *← Оглавление*

11.1 Зрелость как лестница, а не ярлык

Глава 3 дала фундамент — носитель, способный хранить историю и происхождение требований. Но наличие фундамента ещё не говорит, насколько зрело команда им пользуется. Один проект просто держит требования в носителе; другой валидирует их по схеме, гоняет парные тесты и заставляет вторую модель оспаривать первую. Это разные ступени одной лестницы — и эта глава их нумерует.

RENAR определяет **пять уровней зрелости** — `RENAR-1` .. `RENAR-5`, закрытый список. Уровень — это **измеримая характеристика** того, насколько формализовано управление требованиями в проекте. Его важно не путать с **заявлением о соответствии** (соответствие): уровень описывает, *где находится* проект, а соответствие — это процедура, которой он это *доказывает*. Механика заявления — в главе 13.

Каждая следующая ступень добавляет обязательства поверх предыдущей и закрывает свой класс расхождений; читать главу стоит по порядку: §11.4–§11.8 разбирают ступени по одной, §11.10–§11.11 отвечают на «с чего начать» и «как расти».

Граница главы строгая. Она нормирует **только** критерии уровней и переходы между ними. Процедуры заявления о соответствии — глава 13; метрики измерения уровня — глава 12; сами capabilities носителя — глава 3.

11.2 RENAR-M как предметно-специфичная размерность в модели зрелости SENAR

11.2.1 Модель зрелости SENAR (общая зрелость)

SENAR определяет одномерную модель из пяти уровней общей зрелости команды и методологии: `Стихийный → Супервизируемый → Измеримый → Управляемый → Оптимизирующий`. Эта модель оценивает **процессную зрелость команды в целом**, независимо от конкретной процессной области.

11.2.2 RENAR-M как отдельная размерность

RENAR-M — **отдельная размерность** в общей зрелости по SENAR, специализирующая её для процессной области «инженерия требований». Проект характеризуется **парой**:

```
проект ──▶ (SENAR-N, RENAR-M)

где N ∈ {1, 2, 3, 4, 5} — общая SENAR зрелость
    M ∈ {1, 2, 3, 4, 5} — RENAR-уровень управления требованиями
```

Пары `(SENAR-N, RENAR-M)` нормативно независимы: проект на SENAR-4 (Управляемый) может находиться на RENAR-2 (Зафиксированный, Documented) — команда зрелая в SENAR-практиках в целом, но **именно управление требованиями** формализовано слабо. Это допустимый и наблюдаемый сценарий.

11.2.3 Согласованность с SENAR Reference

SENAR Reference допускает предметно-специфичные размерности зрелости (аутентификация, безопасность, наблюдаемость и иные процессные области). RENAR-M — одна из таких размерностей; она не противоречит SENAR и не дублирует его.

В индустрии типична ситуация, когда в одной организации разные процессные области имеют различную зрелость. RENAR-M — нормативная размерность зрелости именно для области «инженерия требований», с критериями только из этой главы и смежных параграфов стандарта.

11.2.4 Соответствие как пара

Манифест соответствия (§13.4) фиксирует **только** RENAR-M (поле `level`). SENAR-N остаётся в области SENAR-соответствия и в RENAR-манифесте не дублируется.

11.3 Закрытый список уровней RENAR-1..RENAR-5

Список из пяти уровней закрыт. Изменение списка возможно только через формальную процедуру изменения стандарта (§13.9.3).

Уровень	Краткое название	Семантика (одной строкой)
`RENAR-1`	Стихийный (Ad-hoc)	Носитель требований существует; артефакты ведутся без формальной схемы и жизненного цикла
`RENAR-2`	Зафиксированный (Documented)	Артефакты имеют базовый frontmatter и хранятся структурно; T3 зафиксировано
`RENAR-3`	Учитываемый (Tracked)	Полная схема frontmatter; статусы жизненного цикла используются; delta-T3 рабочий поток через ADAPT
`RENAR-4`	Верифицируемый (Verified)	100% `approved` имеют `verified-by`; pos/neg парность; QG-2 обеспечивается; AI-происхождение
`RENAR-5`	Оптимизирующий (Optimized)	Состязательный обзор как gate; несколько моделей для `priority: must`; граф знаний; метрики

Промежуточные уровни (`RENAR-3.5`) и локальные для проекта уровни запрещены (§13.9.2). Локальное ужесточение критериев в пределах объявленного уровня разрешено через `declared-stricter` (§13.4.2).

Каждый уровень включает нормативные критерии всех нижестоящих. `RENAR-M` подразумевает выполнение требований `RENAR-1` .. `RENAR-(M-1)`.

11.4 RENAR-1: Стихийный (Ad-hoc)

Первая ступень отвечает на вопрос: **где живут требования?** Ответ — «в носителе, а не в чьей-то голове, трекере задач или переписке». Формальной схемы и жизненного цикла ещё нет, но происхождение уже восстановимо.

11.4.1 Нормативное определение

RENAR-1 — нижний соответствующий уровень. Проект обязан удовлетворять: носитель существует физически (V1–V6 §13.3.2 — абсолютное обязательное положение независимо от уровня); требования ведутся как артефакты внутри носителя (не в трекерах задач или переписке); существует ADAPT для каждого ТЗ (§13.3.3); применяется независимый от носителя язык (§2.5.4).

> ***RENAR-1 ≠ нулевая инфраструктура.*** *«Стихийный» — про отсутствие формальной схемы / жизненного цикла, а не носителя: V1–V6 обязательны на любом уровне. Плоский файловый сервер, Notion, Google Docs, Confluence без immutable history / atomic change / version pin* ***не квалифицируются*** *даже на RENAR-1. Минимум — distributed VCS или document-oriented store с V1–V6 (§3, guide/03–04).*

11.4.2 Что НЕ требуется на RENAR-1

Стандартизированный frontmatter (допустим минимум `id` + `title`); статусы жизненного цикла; ТС как отдельные артефакты (допустимо вести в коде); COVERAGE; нативные для носителя hooks жизненного цикла.

11.4.3 Наблюдаемые сигналы

Артефакт-файлы BR/SR/ADAPT в носителе (не в трекере/чате); на запрос «откуда это требование» ответ через носитель; манифест (§13.4) с `level: RENAR-1`.

11.4.4 Применение QG (обеспечение соблюдения)

QG-0/QG-1/QG-2 нормативно required (§13.3.6), но обеспечение соблюдения через hooks **не обязательно** — проверка человеком вручную по мере необходимости.

11.5 RENAR-2: Зафиксированный (Documented)

RENAR-2 добавляет к существованию **структуру**: базовый frontmatter, ТЗ как неизменяемый артефакт, предсказуемое место для артефактов. Требование можно найти, процитировать и сослаться на него. Машинной проверки ещё нет; дисциплину держат люди.

11.5.1 Нормативное определение

Поверх RENAR-1: каждый BR/SR имеет frontmatter с обязательными полями (§6.5.2, §6.6.2) — минимум без строгой schema-валидации; ТЗ — неизменяемый артефакт (§7.4.2); структурное хранение (логические папки / нативные коллекции); delta-ТЗ — явный артефакт, не устный.

11.5.2 Что НЕ требуется на RENAR-2

CI-валидация frontmatter по схеме; полный жизненный цикл (статусы по желанию); ТС расширения `tc-type`; pos/neg парность ТС.

11.5.3 Наблюдаемые сигналы

Все BR/SR имеют валидный (по минимуму) frontmatter; T3 — один нативный артефакт; delta-T3 как явный change-set (§7.6); манифест с `level: RENAR-2`.

11.5.4 Применение QG (обеспечение соблюдения)

QG-0 (§10.3.1) — процедурный gate (явное утверждение с V6 author + timestamp), без автоматической блокировки hooks.

11.6 RENAR-3: Учитываемый (Tracked)

На RENAR-3 включается **машина**. Frontmatter валидируется по схеме, статусы жизненного цикла работают, delta-T3 проходит через ADAPT. Носитель блокирует ссылку на неутверждённый ADAPT и не даёт реализации сослаться на требование без привязки версии.

11.6.1 Нормативное определение

Поверх RENAR-2: frontmatter валидирован по схеме (reference/02-schemas.md) нативным механизмом (§10.11.1); статусы жизненного цикла реально используются (глава 10); TC существуют для всех BR/SR/SPEC с `priority: must`; COVERAGE авто-генерируется (§9.15); delta-T3 — change-set с анализом влияния (§9.16); reference-validation hook (§10.11.1) блокирует создание артефакта со ссылкой на ADAPT ниже `approved`; реализационный носитель привязывает `verifies[].version` (§9.4, V5).

11.6.2 Наблюдаемые сигналы

Hook валидации frontmatter возвращает структурированный отклик при нарушении схемы; каждый артефакт ∈ { `draft`, `approved`, `verified`, `deprecated`, `obsolete` } (§10.5); COVERAGE обновляется в разумный срок; при delta-T3 архитектор автоматически видит затронутые SR/SPEC/TC; манифест с `level: RENAR-3`.

11.6.3 Применение QG (обеспечение соблюдения)

QG-0 + QG-1 обеспечиваются нативно; QG-2 — частично (проверка `verifies[]` обязательна, pos/neg парность (§9.7) не обязана автоматической).

11.7 RENAR-4: Верифицируемый (Verified)

RENAR-4 — рубеж **доверия к тестам**. Каждое нормативное утверждение покрыто парой pos/neg TC, QG-2 блокирует перевод в `verified` без зелёных тестов на актуальной версии, спот-чек раз в итерацию ловит тесты, подделанные под код.

11.7.1 Нормативное определение

Поверх RENAR-3: 100% артефактов в `approved` имеют `verified-by` со ссылкой на ≥ 1 TC (§9.4); pos/neg парность (§9.7) выполнена для **каждого** нормативного утверждения (single-TC-coverage допустим только для инвариантов с негативной семантикой); QG-2 (§10.3.3) **обеспечивается** — перевод в `verified` блокируется при отсутствии всех `passing` TC на текущей `requirement-version`; все TC автоматизированы (`automation.status: automated`) либо помечены `manual-pending` с дедлайном (§9.5); для `tc-type: ux` — изоляция судьи VLM (§9.13.4); для `tc-type: eval` модель-судья отличается от модели реализации (Decision #8); AI-происхождение во frontmatter (§4.10.1): минимум `generated-by` и `generated-at` обязательны; цитирование источника в теле артефакта (`[TZ-XXX §Y line Z]` или маркер `derived`); hook непрерывной сверки (§2.4.2) по расписанию (не реже раз в неделю); спот-чек 5 случайных passing TC раз в итерацию (§9.14) в audit-trail.

11.7.2 Наблюдаемые сигналы

COVERAGE содержит `verified-by-percent: 100%` для `approved`; попытка перевода BR/SR/SPEC в `verified` без всех passing TC возвращает блокирующую ошибку; выборка 5 случайных артефактов показывает цитирование источника на все нормативные утверждения; манифест с `level: RENAR-4`.

11.7.3 Применение QG (обеспечение соблюдения)

QG-0/QG-1/QG-2 обеспечиваются нативно. QG-3 (Architecture, §10.4.1) — declared, если проект использует ADAPT с двойной подписью (по умолчанию для регулируемых отраслей).

11.8 RENAR-5: Оптимизирующий (Optimized)

RENAR-5 замыкает контур **AI-надёжности**. Вторую модель ставят оспаривать первую, критичные требования генерируются несколькими моделями с обязательным разбором расхождений, частота галлюцинаций удерживается ниже 1%. Стандарт становится системой, которая сама себя проверяет.

11.8.1 Нормативное определение

Поверх RENAR-4: **состязательный обзор** — обязательный gate для `draft → approved` (артефакт проходит рецензирование второй AI-моделью с фиксацией в audit-trail); **multi-model agreement** для `priority: must` (артефакт генерируется ≥ 2 моделями; расхождения помечаются `[multi-model-disagreement]` и обязательны к разбору); **cost/latency budget** на артефакт (`cost-budget` + `latency-budget`; превышение запускает автоматическую декомпозицию); **граф знаний** (reference/05) — первичный поиск AI-агентов; **continuous evaluation** для всех SPEC-AI (§8.5.7); **Hallucination Rate** (глава 12) измеряется и < 1%; **Multi-model Disagreement Rate** (глава 12) измеряется, тренды отслеживаются; возврат улучшений шаблонов в `requirements-library` — стандартная практика.

11.8.2 Наблюдаемые сигналы

Каждый promote-нутый артефакт имеет в audit-trail запись состязательного обзора; для `priority: must` BR — маркер `[multi-model-agreement]` или `[multi-model-disagreement]`; дашборд графа знаний доступен; дашборд Hallucination Rate < 1% на скользящем окне; манифест с `level: RENAR-5`.

11.8.3 Применение QG (обеспечение соблюдения)

Все обязательные gates (QG-0/1/2) обеспечиваются нативно. Состязательный обзор обеспечивается как часть QG-0 — носитель блокирует `draft → approved` без подтверждения. QG-3 declared. QG-4 declared, если проект применяет post-release outcomes (§10.4.2).

11.9 Сравнительная таблица признаков

Всё, что разнесено по пяти секциям выше, в одной матрице. Заполнение: ❌ — не требуется; `частично` — частично; ✓ — нормативно обязательно.

Признак	RENAR-1	RENAR-2	RENAR-3	RENAR-4	RENAR-5
Носитель с V1–V6	✓	✓	✓	✓	✓
ADAPT для каждого T3	✓	✓	✓	✓	✓
Frontmatter стандартизирован	❌	частично	✓	✓	✓
Статусы жизненного цикла используются	❌	частично	✓	✓	✓
Валидация frontmatter по схеме (hook носителя)	❌	❌	✓	✓	✓
TC как полноценный артефакт	❌	❌	частично	✓	✓
Pos/neg парность для каждого утверждения	❌	❌	❌	✓	✓
COVERAGE авто-генерируется	❌	❌	✓	✓	✓
Reference-validation hook	❌	❌	✓	✓	✓
Привязка `verifies[].version` (V5)	❌	❌	✓	✓	✓
QG-0 обеспечивается нативно для носителя	❌	частично	✓	✓	✓
QG-2 обеспечивается нативно для носителя	❌	❌	частично	✓	✓
AI-происхождение во frontmatter	❌	❌	❌	✓	✓

Цитирование источника в теле артефактов	✗	✗	✗	✓	✓
Состязательный обзор как gate	✗	✗	✗	✗	✓
Согласование нескольких моделей для `priority: must`	✗	✗	✗	✗	✓
Бюджет стоимости и задержки на артефакт	✗	✗	✗	✗	✓
Граф знаний как первичный поиск	✗	✗	✗	✗	✓
Непрерывная сверка	✗	✗	✗	✓ (базовая)	✓ (полная)
Непрерывная оценка (SPEC-AI)	✗	✗	✗	✗	✓
Hallucination Rate < 1%	н/п	н/п	н/п	н/п	измеряется и контролируется
Тренд Multi-model Disagreement Rate	н/п	н/п	н/п	н/п	отслеживается

11.10 Минимальный входной уровень (entry-level)

`RENAR-1` — нормативная **нижняя граница** манифеста соответствия. Проект, не выполняющий обязательные положения (§13.3) — в частности, без носителя с V1–V6 или без ADAPT для каждого T3 — **не** имеет уровня RENAR-M вовсе; манифест соответствия для такого проекта не выпускается.

Тип проекта	Рекомендуемый целевой уровень
Короткий эксперимент (spike), < 1 спринта, без контракта	RENAR-2 — достаточно для документирования
Внутренняя автоматизация, 1–3 месяца	RENAR-3 — учитываемый жизненный цикл
Клиентский продукт по договору	RENAR-4 — верифицируемый, с pos/neg парностью
AI-критическая компонента (зависит от eval)	RENAR-5 — обязательно
Регулируемые отрасли (медицина, финтех, госсектор)	RENAR-4 минимум, RENAR-5 рекомендовано

Стандарт допускает **разные уровни в разных проектах** одной организации — требования унифицировать уровень по портфелю нет.

Негативный сценарий: носитель, в котором отсутствует хотя бы одна capability из V1–V6 (§3.2), нормативно не может реализовать никакой RENAR-M — даже `RENAR-1`. Это структурное ограничение, не операционное; манифест такого проекта невалиден (§13.3.2).

11.11 Путь RENAR-1 → RENAR-5

Нормативная последовательность шагов между соседними уровнями. Времена — ожидаемый порядок (для небольшого проекта; это не нормативная гарантия).

11.11.1 RENAR-1 → RENAR-2

1. Создать структурное хранение артефактов в носителе (логические папки или нативные для носителя коллекции).
2. Перенести существующие требования из трекера задач, чата или документов в нативные для носителя артефакты с минимальным frontmatter (`id` , `title`).
3. Зафиксировать ТЗ как неизменяемый артефакт (§7.4.2).
4. Договориться в команде: новые требования — только через носитель, не через трекер задач.

Ожидаемое время: 1–2 недели для небольшого проекта; 1–2 месяца для проекта с большим объёмом накопленных требований.

11.11.2 RENAR-2 → RENAR-3

1. Привести frontmatter всех артефактов к schema (нативный для носителя нормализатор).
2. Включить hook носителя для schema-валидации (блокировать change-set при нарушении).
3. Внедрить жизненный цикл: пройтись по всем артефактам, проставить статусы.
4. Включить reference-validation hook (§10.11.1).
5. Сгенерировать первоначальный авто-генерируемый артефакт COVERAGE (§9.15).
6. Создать ТС для артефактов с `priority: must` .
7. Внедрить привязку `verifies[].version` из реализационного носителя.

Ожидаемое время: 2–4 недели, включая обучение команды.

11.11.3 RENAR-3 → RENAR-4

1. AI-агент проходит по всем артефактам и генерирует пары pos/neg ТС для каждого нормативного утверждения.
2. Реализация ТС в коде / SPEC-runners — параллельно с продуктовой работой; растягивается на 1–2 квартала.
3. Включить hook носителя QG-2 (блокировать перевод в `verified` без всех passing ТС).
4. Внедрить процесс спот-чека (§9.14).
5. Включить hook непрерывной сверки с еженедельным расписанием.
6. Внедрить AI-происхождение во frontmatter (hook носителя блокирует при отсутствии для AI-генерируемых артефактов).
7. Внедрить цитирование источника в теле артефактов (hook носителя блокирует при отсутствии цитаты для нормативных утверждений).

Ожидаемое время: 1–2 квартала.

11.11.4 RENAR-4 → RENAR-5

1. Подключить состязательный обзор второй модели (отличной от первичной; принцип изоляции судьи §9.13.4); включить как обеспечение соблюдения QG-0.

2. Включить генерацию несколькими моделями для артефактов с `priority: must`.
3. Развернуть граф знаний (reference/05).
4. Настроить прицельные метрики Hallucination Rate и Multi-model Disagreement Rate (глава 12).
5. Внедрить бюджет стоимости и задержки с автоматической декомпозицией при превышении.
6. Закрепить практику возврата улучшений шаблонов в `requirements-library`.
7. Непрерывная оценка для всех артефактов `SPEC-AI` (§8.5.7).

Ожидаемое время: 1–2 квартала после RENAR-4.

11.11.5 Обратные переходы (понижение уровня, downgrade)

Нормативное понижение уровня (§13.8.2) — выпуск новой версии манифеста с уровнем ниже текущего — допустимо при наличии формального обоснования (например, вывод из эксплуатации AI-критической компоненты, упрощение носителя). Понижение обязано сопровождаться записью в audit-trail; скрытие понижения — нарушение обязательного положения (§13.3.1).

11.12 Связь с SENAR ADR (Adversarial Detection Rate)

SENAR §9 определяет ADR — Adversarial Detection Rate — как общую метрику способности процесса обнаружить ошибки до выхода в промышленную эксплуатацию. RENAR-M определяет, на каких уровнях существует нормативная инфраструктура состязательного обзора. RENAR-специфичный подкласс ADR для зоны требований — **ACR** (Adversarial Catch Rate, §12.3.6).

RENAR-M	Нормативная инфраструктура состязательного обзора	Измеримость ADR / ACR
RENAR-1, RENAR-2	Отсутствует (§11.4, §11.5)	н/п
RENAR-3	Нормативно отсутствует; команда может ввести состязательный обзор как локальное ужесточение (`declared-stricter`, §13.4.2)	опционально
RENAR-4	Pos/neg парность TC (§9.7) — структурный прокси ADR; состязательный обзор артефактов нормативно не требуется	опционально (покрытие pos/neg измеримо как прокси)
RENAR-5	Состязательный обзор как обязательный gate (§11.8.1) + согласование нескольких моделей для `priority: must`	нормативно измеримо (ACR, §12.3.6)

Зрелость RENAR-M связана с SENAR-метрикой ADR **непрерывно**, без дублирования: SENAR ADR задаёт понятие метрики; уровень RENAR-M определяет, какие состязательные циклы нормативны; ACR (§12.3.6) — предметно-специфичная метрика для зоны требований.

11.13 Связь с другими главами

Модель зрелости — это карта, по которой расставлены требования всех остальных глав: каждая глава «включается» на своём уровне. Таблица ниже показывает, где именно.

Глава	Связь

02 Положение в типологии методологий	§2.3 инверсия источника истины + §2.5 независимое от носителя версионирование — обязательные положения независимо от уровня; §2.4 четыре отстройки — наблюдаются на всех уровнях, начиная с RENAR-2
06 Иерархия требований	frontmatter артефактов и обязательные поля проверяются на RENAR-3+; иерархия BR/SR/TR — на всех уровнях
07 ADAPT	ADAPT для каждого T3 — обязательное положение независимо от уровня (§11.4.1); двойная подпись QG-3 — declared на RENAR-4+
08 Specifications	Закрытый список 9 типов SPEC — обязателен; полное type-specific покрытие на RENAR-4+; непрерывная оценка SPEC-AI на RENAR-5
09 Test cases	TC для `priority: must` на RENAR-3; pos/neg парность на RENAR-4+; процесс спот-чека на RENAR-4+
10 Жизненный цикл и QG	QG-0/QG-1/QG-2 объявлены required на всех уровнях; обеспечение соблюдения носителем постепенное — частичное на RENAR-2, полное на RENAR-4+; QG-3 declared на RENAR-4+, если используется ADAPT
03 Версионирование носителя	V1–V6 — абсолютное обязательное положение для любого уровня; носитель без V1–V6 не имеет уровня RENAR-M (§11.10)
12 Метрики	метрики Hallucination Rate / Multi-model Disagreement Rate / Defect Escape Rate измеряются на RENAR-4+; конкретные определения — в главе 12
13 Соответствие	§13.2 ссылается на эту главу за критериями каждого уровня; §13.5 self-assessment использует checklists §§11.4–12.8 (по одной секции на уровень); §13.9 политика закрытого списка применяется к закрытому списку RENAR-1..5 (§11.3)

12. Метрики

> ***Часть RENAR Standard v1.0-draft*** · *← Оглавление*

12.1 Что измерять в требованиях

Общие метрики SENAR (§9) покажут, что команда быстра и тесты зелёные. Но они не заметят, как AI-агент тихо вписал в SR пункт, которого не было ни в ТЗ, ни в ADAPT, — пока он не всплывёт на приёмке спором с клиентом. «Процесс в целом» здоров, а работа с требованиями — нет. Поэтому RENAR добавляет **десять метрик, которые смотрят именно на требования**: как часто AI-агент придумывает пункт без источника (Hallucination Rate), ловят ли парные тесты реальные дефекты, как быстро дельта-ТЗ доходит до кода. Это надстройка над SENAR §9, а не замена.

Список метрик закрыт; для каждой задаются формула, цель по уровню зрелости (глава 11) и источник данных. Метрики собираются нативно для носителя через V1–V6 (глава 3); как именно рисовать дашборды — вопрос реализации, вынесенный в `guide/`. Глава не дублирует SENAR §9, а специализирует его, и не касается ROI или ценообразования — это не индикаторы процесса, а бизнес-эффекты (§12.5).

12.2 Связь с SENAR §9

SENAR §9 определяет десять метрик общего процесса: Throughput, Lead Time, FPSR (First-Pass Success Rate), DER (Defect Escape Rate), KCR (Knowledge Capture Rate), Cost Predictability, Cost-per-task, MIR (Memory Integrity Rate), Cycle Time, ADR (Adversarial Detection Rate).

RENAR §12 **не** редактирует и **не** заменяет эти метрики. REQ-специфичные метрики §12.3:

- **Уточняют** SENAR метрику для фазы требований (например, RDLT уточняет SENAR Lead Time на фазе требований).
- **Добавляют** наблюдения, специфичные для инженерии требований и не покрываемые SENAR §9 (Hallucination Rate, Multi-model Disagreement Rate).

Полный mapping — §12.7.

Закрытый список REQ-метрик (§12.3) сохраняется в рамках RENAR; SENAR §9 — отдельный закрытый список общих метрик. Изменение любого из двух списков — формально независимые процедуры изменения соответствующих стандартов.

12.3 Закрытый список REQ-специфичных метрик

Закрытый список из десяти REQ-метрик. Изменение списка — только через формальную процедуру изменения стандарта (§13.9.3); общая политика закрытого списка и master-индекс — §1.7.5.

12.3.1 RDLT — Requirement Decomposition Lead Time

Время от регистрации ТЗ в носителе до состояния «все BR/SR (parent цепочка из этого ТЗ) → `approved`, готовы для QG-0 (§10.3.1)». **Формула:** `RDLT = timestamp(last BR/SR →`

`approved) – timestamp(TZ registered)`. Измеряется в часах/днях. **Источник:** журнал аудита promote-transitions (§10.13). **Связь с SENAR:** уточнение SENAR `Lead Time` для фазы требований. **Цели:** RENAR-3 < 1 неделя на 50-страничный ТЗ; RENAR-4 < 2 дня; RENAR-5 < 4 часа.

12.3.2 Requirement-to-Task Latency

Время от promote-transition SR → `approved` до создания первой TR со ссылкой `implements: SR-N`. **Формула:** `Latency = timestamp(first TR.created) – timestamp(SR → approved)`. Часы. **Источник:** журнал аудита носителя + межносительные ссылки реализационного носителя. **Связь с SENAR:** уточнение SENAR `Cycle Time` для пары «requirement → executable task». **Цели:** RENAR-3 < 3 дня; RENAR-4 < 1 день; RENAR-5 < 1 час (auto-create TR после approval).

12.3.3 Hallucination Rate

Процент нормативных утверждений в AI-генерируемом артефакте (BR/SR/SPEC), которые не traceable к source (ТЗ/ADAPT/другой нормативный артефакт). Source citation проверяется нативным citation parser (§13.3.1, RENAR-4 обязательно). **Формула:** `assertions_without_valid_citation / total_normative_assertions × 100%`. Per артефакт; агрегируется per project. **Источник:** citation parser (AST или regex по inline references `[TZ-XXX §Y]` / `[ADAPT-NNN §Z]`). **Связь с SENAR:** новая метрика; соответствует ISO/IEC 5338 traceability для артефактов, сгенерированных AI. **Цели:** RENAR-1..3 n/a; RENAR-4 ≤ 5%; RENAR-5 ≤ 1%.

Отрицательный сценарий (триггер потери соответствия): Hallucination Rate > 5% на RENAR-4 проекте — нормативный триггер потери соответствия (§13.8.1); устранить через план восстановления релиза либо понижение до RENAR-3.

12.3.4 Multi-model Disagreement Rate

Процент артефактов с `priority: must`, где две (или более) генерирующие AI-модели произвели нормативные утверждения с расхождением выше порога (по умолчанию embedding similarity < 85%; порог фиксируется в манифесте `declared-stricter`). **Формула:** `BRs_with_high_disagreement / total_must_BRs × 100%`. **Источник:** multi-model runs log; embedding-similarity computed offline по парам «model A vs B». **Связь с SENAR:** новая метрика. **Цели:** RENAR-1..4 n/a; RENAR-5 tracked, базовые значения по проекту в первом квартале. **Интерпретация:** высокое значение — индикатор слабого prompt-engineering или сложной предметной области; требует внимания, но само по себе не является негативным показателем.

12.3.5 DRA — Dispute Rate at Acceptance

Процент BR/SR, по которым на этапе QG-4 (§10.4.2) клиент заявил несогласие с интерпретацией или покрытием. **Формула:** `disputed_BRs_at_QG4 / total_BRs_in_release × 100%`. **Источник:** журнал аудита QG-4 — `gate-id: QG-4` event с `result: disputed`. **Связь с SENAR:** уточнение `DER (Defect Escape Rate)` для requirements. **Применимость:** только при QG-4 в манифесте; при QG-4 = `absent` (§13.3.6) — не измеряется. **Цели:** RENAR-3 ≤ 10%; RENAR-4 ≤ 5%; RENAR-5 ≤ 2%.

12.3.6 ACR — Adversarial Catch Rate

Процент артефактов (BR/SR/SPEC), где AI-критик (другая модель; обязательна на RENAR-5 per §11.8.1) нашёл ≥ 1 high-severity issue до QG-0. **Формула:** `artifacts_with_critic_high_findings / total_reviewed_by_critic × 100%`. **Источник:** журнал аудита critic-runs; severity (`high` / `medium` / `low`) в critic output. **Связь с SENAR:** подкласс `ADR (Adversarial Detection Rate)` для requirements. **Цели:** RENAR-1..3 n/a; RENAR-4 optional (если `declared-stricter` — базовый уровень 20–30%; значения < 20% — индикатор слабого критика или дублирования primary); RENAR-5 tracked normatively, рост ACR требует внимания к качеству промптов.

12.3.7 Test-spec Drift Rate

Процент TC в статусе `passing` (§10.9.1), у которых `last-run.requirement-version` (§9.12) отличается от текущей `version` верифицируемого артефакта (`verifies[]`). **Формула:** `stale_passing_TCs / total_passing_TCs × 100%` (stale = `last-run.requirement-version` < текущей). **Источник:** COVERAGE-artifact (§9.15). **Связь с SENAR:** новая метрика. **Цели:** RENAR-1..3 n/a; RENAR-4 ≤ 5%; RENAR-5 ≤ 1% (auto-rerun на delta-ADAPT).

12.3.8 Coverage Velocity

Темп перехода `approved` → `verified` (§10.5) за единицу времени (default итерация; носитель может определить другой интервал в манифесте). **Формула:** `(verified_count(end) − verified_count(start)) / approved_count(start) × 100%`. **Источник:** COVERAGE-artifact history (§9.15). **Связь с SENAR:** уточнение `Throughput` для requirements. **Цели:** RENAR-3 ≥ 30%/итерация; RENAR-4 ≥ 50%/итерация; RENAR-5 ≥ 70%/итерация.

12.3.9 Cost per Approved Requirement

Стоимость AI-генерации (input tokens + output tokens × tariff) приведённая к одному артефакту в статусе `approved`, включая отвергнутые версии (остаются в носителе за счёт V1 immutable history). **Формула:** `total_AI_tokens_cost(period) / count(artifacts_approved_in_period)`. Валюта проекта. **Источник:** `ai-provenance.cost-budget` + `ai-provenance.cost-actual` поля frontmatter (§11.7.1). **Связь с SENAR:** уточнение `Cost-per-task` для requirements. **Цели:** RENAR-1..3 n/a; RENAR-4 tracked, базовые значения по проекту; RENAR-5 tracked + year-over-year снижение либо обоснование.

12.3.10 Reconciliation Findings per Week

Количество issues, обнаруженных reconciliation-агентом (§2.4.2 continuous reconciliation) за неделю и зарегистрированных как backward findings в delta-ADAPT либо direct change-set requirement. **Формула:** `count(reconciliation_findings_registered) / weeks_in_period`. **Источник:** журнал аудита reconciliation runs + backward findings list ADAPT (§7.4.5). **Связь с SENAR:** новая метрика. **Цели:** RENAR-1..3 n/a; RENAR-4 tracked > 0 (ноль = reconciliation hook не работает

либо потеря V5); RENAR-5 trending **down** в долгосрочной перспективе (зрелый процесс не порождает новых drift).

12.4 Сводная таблица целей по уровням

Закрытый список 10 метрик из §12.3 — целевые значения по применимым уровням.

Метрика	RENAR-3	RENAR-4	RENAR-5	Источник данных
RDLT (Decomposition Lead Time)	< 1 неделя	< 2 дня	< 4 часа	promote-transitions журнал аудита
Requirement-to-Task Latency	< 3 дня	< 1 день	< 1 час	журнал аудита + межносительные refs
Hallucination Rate	n/a	≤ 5%	≤ 1%	citation parser
Multi-model Disagreement Rate	n/a	n/a	tracked	multi-model runs log
DRA (Dispute Rate at Acceptance)	≤ 10%	≤ 5%	≤ 2%	QG-4 журнал аудита (если QG-4 declared)
ACR (Adversarial Catch Rate)	n/a	optional (если critic declared-stricter)	tracked normatively	critic-runs журнал аудита
Test-spec Drift Rate	n/a	≤ 5%	≤ 1%	COVERAGE-artifact
Coverage Velocity	≥ 30%/итерация	≥ 50%/итерация	≥ 70%/итерация	COVERAGE history
Cost per Approved Requirement	n/a	tracked	tracked + optimized	ai-provenance fields
Reconciliation Findings/Week	n/a	tracked (> 0)	trending down	reconciliation журнал аудита

Конкретное нативное для носителя представление этих метрик в dashboards — специфично для носителя и выносится в `guide/`.

> *Целевые значения для RENAR-4 / RENAR-5 — **provisional**: заданы как нормативные ориентиры направления, но подлежат калибровке по field-data в версии v1.1 (см. guide/07 §8.1). Это направляющие пороги, а не статистически валидированные значения.*

12.5 Бизнес-результаты (Business outcomes)

Шесть нормативных эффектов внедрения RENAR, ожидаемых на уровнях RENAR-3 и выше. Outcomes являются **нормативными ожиданиями стандарта**, не индикаторами процесса; их измерение — специфично для носителя и не обязательно (хотя §12.3 метрики косвенно их фиксируют).

> *ROI / cost-of-adoption — **ненормативная** тема и в нормативную главу не входит. Облегчённая **иллюстративная** (не гарантированная) модель стоимости и выгоды внедрения для лица,*

> *принимающего решение, — в guide/02-transition-guide.*

12.5.1 Результат 1 — Сокращение времени декомпозиции T3

Измеряется через §12.3.1 RDLT. Ожидаемое сокращение от базового уровня: порядок 5–10× на RENAR-4, 20–50× на RENAR-5.

12.5.2 Результат 2 — Снижение частоты споров на приёмке

Измеряется через §12.3.5 DRA. Ожидаемое снижение: с 15–30% до ≤ 5% на RENAR-4, ≤ 2% на RENAR-5.

12.5.3 Результат 3 — Аудиторская готовность

Журнал аудита событий (§10.13) + AI-provenance во frontmatter (§11.7.1) обеспечивают compliance audit без отдельной подготовительной работы. Применимо для regulated industries (медицина, финтех, госсектор).

12.5.4 Результат 4 — Снижение стоимости работы с delta-T3

Impact analysis (§9.16) + reverse-эволюция верификации (§10.5.4) автоматизируют обработку delta-T3. Ожидаемое сокращение участия человека: с десятков часов до часа на delta.

12.5.5 Результат 5 — Снижение потери знаний при смене команды

V6 author + timestamp (§3.3.6) фиксирует автора всех артефактов; инверсия источника истины (§2.3.1) переносит знания из головы в носитель. Ожидаемое сокращение onboarding: с недель до дней.

12.5.6 Результат 6 — Стандарт как продаваемый продукт

При RENAR-4/5 нативная для носителя реализация может быть лицензирована партнёрам как actionable продукт. Структурное следствие formal standard, не процессная метрика.

12.6 Независимый от носителя сбор метрик

12.6.1 Нормативные требования

Носитель, реализующий RENAR на уровне RENAR-4+, обязан обеспечить **автоматический сбор** §12.3 метрик через:

Источник	Capabilities	Accessible
Журнал аудита событий (§10.13)	V1 + V6	gate-passage events с timestamps, artifact-version, actor
COVERAGE-artifact (§9.15)	V5 + V1	counts approved/verified/total; pos/neg coverage; stale-rate
AI-provenance frontmatter fields	V6	cost-budget, cost-actual, generated-by, generated-at
Reconciliation журнал аудита	V1 + V6	reconciliation runs + ID списка находок

Critic-runs журнал аудита (RENAR-5)	V1 + V6	critic runs + severity classifications

Носитель без доступа к любому из источников выше **не** может реализовать RENAR-4/5 (§11.7.1, §11.8.1).

12.6.2 Нативные для носителя панели мониторинга (dashboards)

Формат (UI/CLI/report-generation) специфичен для носителя; стандарт не нормирует визуализацию. Носитель обязан экспортировать метрики в machine-readable формате для внешнего аудита (§13.6 third-party assessment). Шаблоны dashboard — `guide/` .

12.6.3 Агрегация по периодам (period aggregation)

Per артефакт — Hallucination Rate, Cost per Approved Requirement; per period (sprint/неделя/месяц) — Coverage Velocity, Reconciliation Findings/Week, ACR; per release — DRA; continuous trending — Multi-model Disagreement Rate, Test-spec Drift Rate. Period boundaries фиксируются в манифесте (§13.4.2) `declared-stricter` или принимаются по умолчанию.

12.7 Сопоставление с метриками SENAR (mapping)

Полное сопоставление десяти метрик SENAR §9 с REQ-уточнениями из §12.3.

SENAR метрика (§9)	REQ-уточнение из §12.3
Throughput	+ Coverage Velocity (§12.3.8) — на уровне требований
Lead Time	+ RDLT (§12.3.1) — для requirements phase
FPSR (First-Pass Success Rate)	+ REQ-FPSR (доля артефактов, прошедших QG-0 без переделки) — производное, не отдельная метрика §12.3
DER (Defect Escape Rate)	+ DRA (§12.3.5) — defects на приёмке
KCR (Knowledge Capture Rate)	(используется как есть; косвенно усиливается через §12.5.5 результата)
Cost Predictability	+ variance Cost per Approved Requirement (§12.3.9)
Cost-per-task	+ Cost per Approved Requirement (§12.3.9) — для requirements phase
MIR (Memory Integrity Rate)	(используется как есть; усиливается через V1 + V6 на RENAR-4+)
Cycle Time	+ RDLT (§12.3.1) + Requirement-to-Task Latency (§12.3.2) — оба внутри SENAR Cycle Time
ADR (Adversarial Detection Rate)	+ ACR (§12.3.6) — состязательный для requirements зоны

Метрики, **не имеющие** SENAR аналога (новые в RENAR):

- Hallucination Rate (§12.3.3) — специфична для AI-generated artifacts.
- Multi-model Disagreement Rate (§12.3.4) — специфична для multi-model генерации.

- Test-spec Drift Rate (§12.3.7) — специфика requirement-version pinning V5.
- Reconciliation Findings per Week (§12.3.10) — специфика continuous reconciliation.

12.8 Связь с другими главами

Глава	Связь
02 Положение в типологии методологий	§2.3 инверсия источника истины + §2.4.2 continuous reconciliation — фундамент для §12.3.10 Reconciliation Findings
07 ADAPT	§7.4.5 backward findings — input для §12.3.10; delta-ADAPT — измеряется через §12.3.5 DRA
08 Specifications	§8.5.7 SPEC-AI continuous evaluation — связана с §12.3.4 Multi-model Disagreement Rate
09 Test cases	§9.12 last-run — input для §12.3.7 Drift Rate; §9.15 COVERAGE — источник для §12.3.8 Velocity и §12.3.7
10 Жизненный цикл и QG	§10.13 журнал аудита событий — основа для всех §12.3 метрик; §10.4.2 QG-4 — gate, на котором фиксируется §12.3.5 DRA
03 Версионирование носителя	V1 + V5 + V6 — capabilities обязательные для нативного для носителя сбора §12.3 метрик (§12.6.1)
11 Модель зрелости	§12.3 цели по уровням RENAR-3/4/5 — конкретизация уровневых критериев из §11.4–§11.8
13 Соответствие	§12.3 метрики — вход для §13.5 самооценки; превышение порогов (например, Hallucination Rate > 5% на RENAR-4) — триггер потери соответствия §13.8.1

13. Соответствие

> ***Часть RENAR Standard v1.0-draft*** *· ← Оглавление*

> ***Плотная глава:*** *начните с reference/08 kit; MVR↔§13.3 bijection — там же §1.*

13.1 Как доказать соответствие

Сказать «у нас всё по RENAR» легко — доказать труднее. Эта глава о том, как проект делает проверяемое заявление: «мы реализуем RENAR на уровне `RENAR-3`» — и подкрепляет его так, чтобы внешний оценщик мог перепроверить. Заявление — не лозунг, а подписанный манифест со ссылками на свидетельства в носителе: вот уровень, вот доказательная база по каждому обязательному положению, вот кто и когда это подтвердил.

Глава нормирует три вопроса. **Кто** вправе заявить — архитектор проекта (самооценка) или независимый оценщик (третья сторона). **На каком основании** — закрытый список уровней RENAR-1..5 плюс универсальные положения, обязательные для любого уровня. **Как заявление живёт во времени** — повторная оценка по расписанию, а при нарушении — потеря соответствия. Если заявление перестаёт быть правдой, это регистрируемое событие, а не тихое умолчание.

Сами уровни даны здесь короткой семантической сводкой: полные критерии RENAR-1..5 — в главе 11, а нативные для носителя механизмы проверок — в главе 3 и `guide/06-compliance.md`. Здесь — правила самого заявления.

13.2 Закрытый список уровней RENAR

Закрытый список из пяти уровней зрелости. Полные критерии — глава 11; ниже — нормативная **семантическая сводка** для заявления о соответствии.

Уровень	Краткая семантика	Статус соответствия
`RENAR-1`	Ad-hoc (стихийный уровень): артефакты требований ведутся в носителе с V1–V6 (§13.3.2); без формальной `frontmatter`-схемы, без статусов жизненного цикла	Минимальный входной уровень (§13.2.1)
`RENAR-2`	Documented (зафиксированный артефактами): носитель требований существует; артефакты имеют базовый `frontmatter`; ТЗ зафиксировано	Соответствие декларируем (§13.2.2)
`RENAR-3`	Tracked (ведётся жизненный цикл и учёт связей): полная `frontmatter`-схема; статусы жизненного цикла используются; delta-ТЗ workflow через ADAPT	Соответствие декларируем (§13.2.2)
`RENAR-4`	Verified (верифицируемый): 100% `approved` артефактов имеют `verified-by`; pos/neg парность (§9.7); QG-2 обеспечивается; AI-происхождение	Соответствие декларируем (§13.2.2)
`RENAR-5`	Optimized (оптимизирующий): состязательный критик; multi-model генерация; knowledge graph; Hallucination Rate measured	Соответствие декларируем (§13.2.2)

13.2.1 RENAR-1 как минимальный входной уровень (entry-level)

`RENAR-1` — нормативный входной уровень. Проект, в котором отсутствует носитель требований (требования живут только в ticket-системах или личной переписке), **не** заявляется как `RENAR-1`; заявление соответствия стандарту начинается с фактического наличия носителя требований.

`RENAR-1` фиксируется в манифесте соответствия только если проект явно декларирует начало пути в RENAR; для проектов без намерения внедрять RENAR манифест соответствия не требуется.

13.2.2 Закрытость списка

Новые уровни (`RENAR-6` , `RENAR-N+`) **не** создаются локально на уровне проекта. Изменение списка уровней — только через формальную процедуру изменения стандарта (§13.9).

Реализация может ужесточить требования относительно уровня (declare-stricter — см. §10.10.2); это не меняет уровень и не выводит проект за пределы закрытого списка.

13.3 Обязательные положения (обязательные положения), универсальные для всех уровней

Нормативные положения, обязательные для **любого** заявления о соответствии независимо от объявленного уровня (включая `RENAR-1`). Нарушение хотя бы одного — отсутствие соответствия стандарту RENAR в целом.

13.3.1 Инверсия источника истины (инверсия источника истины)

Реализация обязана соблюдать **инверсию источника истины** (§2.3): иерархия артефактов требований — источник истины о поведении системы; код — производный артефакт реализации. Эквивалентные нарушения: reverse-engineering поведения из кода в SR без обоснования bug-fix (§2.3.3 (1)); молчаливая адаптация SR под наблюдаемое поведение кода (§2.3.3 (4)).

13.3.2 Возможности носителя V1–V6

Носитель проекта обязан удовлетворять возможностям V1–V6 (глава 3 §3.3):

Возможность	Статус для соответствия
V1 — неизменяемая история	**Обязательно абсолютно**: без V1 невозможен журнал аудита и V5
V2 — атомарная единица изменения	**Обязательно абсолютно**: без V2 невозможна согласованность delta-ADAPT
V3 — сравнение различий и рецензирование	**Обязательно абсолютно**: без V3 нет gates, нет утверждения (§10.11.2)
V4 — ветвление / набор изменений	**Обязательно абсолютно**: без V4 неотделимы WIP и источник истины (§10.11.2)
V5 — сквозная фиксация версии между носителями	**Обязательно абсолютно**: без V5 невозможен `verifies[].version` и QG-2 (§10.3.3)

V6 — автор и отметка времени	**Обязательно абсолютно**: без V6 невозможна двойная подпись ADAPT, AI-происхождение (§10.11.2)

Негативный сценарий: носитель, в котором отсутствует хотя бы одна возможность из V1–V6, нормативно **не** может реализовать никакой RENAR-N — включая `RENAR-1`. Это структурное ограничение (§3.2), не операционное.

13.3.3 Reactive ADAPT

ADAPT — реактивный артефакт (§7.4.1). Каждое ТЗ обязано пройти состязательный обзор (§7.10.2) с зафиксированным в носителе вердиктом (V6 author + timestamp). Дальнейшее зависит от вердикта:

Вердикт состязательного рецензента	Требование к ADAPT	Требование к `source` для BR/SR/SPEC
«findings present» — обнаружены backward findings, term mapping clarification или scope clarification	ADAPT обязателен в статусе `approved`. Двойная подпись (Архитектор + представитель Клиента, §7.5). Lifecycle §7.4.5.	`source.adapt` mandatory; `source.tz-section` mandatory
«no findings, no clarifications» — конвертация ТЗ → RENAR однозначна	ADAPT не создаётся	`source.tz-section` mandatory; `source.adversarial-review-ref` mandatory (свидетельство вердикта)

Делегирование на состязательного рецензента — единственный допустимый способ задекларировать «ADAPT не нужен». Создание BR / SR / SPEC из ТЗ без зафиксированного вердикта — нарушение стандарта; hooks (§10.11.1 `adapt-applicability validation`) обязаны блокировать такие артефакты.

При наличии delta-ТЗ — то же правило (§7.6): delta-ADAPT создаётся по факту находок состязательного обзора delta-ТЗ.

Множественность и стадийная независимость. Триггер ADAPT стадийно-независим: вердикт выносится не только при импорте ТЗ, но и на стадиях деривации (BR → SR → SPEC → TC, §7.4.1.1). На одно ТЗ приходится **ноль или более** корневых ADAPT (кардинальность 0..N, MVR-3, §7.4.1.4); каждый фиксирует `trigger-stage`. Множественность соответствует стандарту и не ослабляет провенанс: каждый BR / SR / SPEC ссылается ровно на один ADAPT либо на `source.tz-section`.

Дезавуирование (`supersession`). Approved/frozen ADAPT может быть дезавуирован дезавуирующим ADAPT (§7.6.4). Соответствующее стандарту дезавуирование обязано: (1) сохранять дезавуируемый ADAPT в терминальном `superseded` (неизменяемо, V1) — не удалять; (2) при контрактном итоге — нести подпись клиента на дезавуирующем ADAPT; (3) перенаправить или перевывести все производные так, чтобы не осталось висячих `source.adapt` на `superseded` (§6.10.3). Отдельный QG не требуется — используется QG-3 (§10.8.5).

Negative scenarios (несоответствующие):

- Манифест объявляет соответствие, но BR/SR/SPEC производятся из ТЗ без `source.tz-section` и без `source.adapt` — нарушение обязательного происхождения.

- Создание BR/SR/SPEC с пропущенным `source.adapt` **без** свидетельства `source.adversarial-review-ref` — нарушение §7.4.1.3 «запрет молчаливого skip».
- ADAPT создан при вердикте «no findings» (есть свидетельство) — противоречие: вердикт заявляет, что ADAPT не нужен, но ADAPT присутствует.
- ADAPT в статусе `superseded` удалён из носителя — нарушение неизменяемой истории (V1).
- Висячая ссылка `source.adapt` на `superseded` ADAPT (производное не перенаправлено и не пере-выведено) — невалидная цепочка прослеживаемости.
- Дезавуирование решения с контрактным итогом без подписи клиента на дезавуирующем ADAPT — односторонняя отмена согласованного с клиентом, нарушение §7.6.4.

13.3.4 Закрытый список 9 типов SPEC

Тип SPEC обязан принадлежать закрытому списку девяти типов (§8.3): `SPEC-ARCH`, `SPEC-API`, `SPEC-DATA`, `SPEC-INT`, `SPEC-PROC`, `SPEC-UI`, `SPEC-AI`, `SPEC-SEC`, `SPEC-OPS`.

Проект не имеет права создавать новые типы SPEC локально. Изменение списка возможно только через формальную процедуру изменения стандарта (§8.3.1, §13.9).

13.3.5 TC pos/neg парность для нормативных утверждений

Каждое нормативное утверждение верифицируемого артефакта (BR / SR / SPEC / TR), охватываемое хотя бы одним TC, обязано иметь парный negative TC (§9.7). Single-TC-coverage допускается только в одном случае: утверждение само описывает negative invariant (например, `SPEC-SEC` STRIDE-категория).

QG-2 (§10.3.3) обязан блокировать promote артефакта в `verified` при отсутствии хотя бы одного парного negative TC.

13.3.6 Закрытый список Quality Gates

Закрытый список Quality Gates (§10.3, §10.4):

Gate	Статус в манифесте соответствия
QG-0 — Approval	**Обязательно required**
QG-1 — Implementation	**Обязательно required**
QG-2 — Verification	**Обязательно required**
QG-3 — Architecture (опц.)	`declared` (поддерживается) или `absent`; для проектов с обязательным ADAPT — фактически всегда `declared`, поскольку утверждение ADAPT оперирует двойной подписью QG-3 (§10.4.1)
QG-4 — Acceptance (опц.)	`declared` или `absent`; при `absent` терминальный статус артефакта — `verified` (без `accepted`)

Локальное создание новых типов gate запрещено (§10.10.2). Локальное ужесточение предусловий канонического gate разрешено (declare-stricter); локальное ослабление — запрещено.

13.3.7 Закрытые списки backward findings и SPEC-декомпозиций

- Список категорий backward findings в ADAPT закрыт (§7.4.4) — 7 категорий; добавление новых — формальная процедура изменения стандарта (§13.9).
- Закрытый список типов SPEC дополнительно фиксирует §13.3.4.
- Закрытый список состояний жизненного цикла артефактов (глава 10) — не расширяется локально на уровне проекта.

13.3.8 Межуровневая прослеживаемость подсистем (`implements` -edge)

Для сценария «подсистема — самостоятельный продукт» (§6.8.2) реализация обязана обеспечить машиночитаемую цепочку прослеживаемости `BR (подсистема) → BR (система)` через типизированное ребро `implements[]` (§6.5.2):

Уровень соответствия	Требование
v1.0: `recommended`	Если `BR.level = subsystem` И родительская система имеет ≥1 approved BR — наличие `implements[]` рекомендовано. Отсутствие — допустимо, но требует обоснования в разделе `Контекст` со ссылкой на ADAPT§.
v1.1+: `mandatory`	Тот же триггер делает `implements[]` обязательным. Отсутствие при выполнении триггера — несоответствие.

В обеих версиях носитель обязан реализовать точку контроля `implements` -edge validation из §10.11.1 (target существует и в `approved` +, нет циклов, deprecated target → warning, `implements[]` не на `level: system`).

Негативный сценарий: проект объявляет соответствие `RENAR-N` с `BR.level = subsystem` и approved родительским BR в той же системе, но без `implements[]` и без обоснования в Контексте — несоответствующий начиная с v1.1; на v1.0 — прогнозируемо-несоответствующий при upgrade (§13.9 руководство по миграции).

13.4 Манифест соответствия

13.4.1 Расположение и формат

Манифест соответствия хранится в корне носителя требований проекта под именем `RENAR-CONFORMANCE.yaml` . Формат — YAML 1.2; альтернативная сериализация (`.json`) допустима как **дополнительный** артефакт, не замена.

Манифест является **неизменяемым** в смысле V1: каждое заявление о соответствии создаёт новую версию манифеста (`manifest-version` инкрементируется); предыдущие версии не удаляются, остаются в носителе как журнал аудита. Замена манифеста через `replaced-by` указывает на новую версию.

13.4.2 Обязательные поля

```yaml
# RENAR Conformance Manifest schema (mandatory fields, v1.0)
renar-version: "1.0"           # версия стандарта RENAR, к которой заявляется
conformance
senar-version: "1.0"           # версия SENAR, на которую опирается реализация
(обязательно, MVR-7)
manifest-version: 3            # инкрементируется при каждом обновлении; не пере-
используется
manifest-id: "CFM-2026-001"    # стабильный нативный для носителя идентификатор
(V1)

# Уровень conformance
level: "RENAR-3"               # из closed list §13.2 (RENAR-1..RENAR-5)
level-target: "RENAR-4"        # опционально: следующий целевой уровень (для path
planning)

# Assessment metadata
assessment-mode: "self"        # self | third-party
assessment-date: "2026-05-13"  # ISO-8601 даты завершения текущей оценки
assessor:
  id: "architect-andrey-y"     # V6 author identifier
  role: "architect"            # роль из §13.5 (architect | authorized-role-
holder | external-assessor)
  signature-ref: "<нативный для носителя pointer на signature event>"
next-assessment-due: "2026-08-13"  # §13.7

# Mandatory clauses подтверждение (§13.3)
mandatory-clauses-confirmed:
  sot-inversion: true                              # §13.3.1
  substrate-v1-v6: { v1: true, v2: true, v3: true, v4: true, v5: true, v6: true }
# §13.3.2
  adapt-per-tz: true                               # §13.3.3
  spec-types-closed-list: true                     # §13.3.4
  tc-pos-neg-pairing: true                         # §13.3.5
  quality-gates-closed-list: true                  # §13.3.6
  closed-lists-backward-findings: true             # §13.3.7

# Quality gates declaration (§10.4.3)
quality-gates:
  qg-0: required               # required (обязательно)
  qg-1: required
  qg-2: required
  qg-3: declared               # required | declared | absent
  qg-4: absent

# Декларация возможностей носителя (§13.3.2)
substrate-capabilities:
  v1-immutable-history: declared
  v2-atomic-change-unit: declared
  v3-diff-review: declared
  v4-branching: declared
```

```
  v5-version-pin: declared
  v6-author-timestamp: declared
  substrate-id: "<нативный для носителя pointer на guide/03..06>"   # cross-ref
на guide

# Spec types support (§13.3.4)
spec-types-supported: ["SPEC-ARCH", "SPEC-API", "SPEC-DATA", "SPEC-INT",
                       "SPEC-PROC", "SPEC-UI", "SPEC-AI", "SPEC-SEC", "SPEC-OPS"]
# Все 9 типов обязательны как minimum-supported; declaration «type X не
используется в проекте» допустима,
# но НЕ может быть declaration «type X не поддерживается нативно для носителя».

# Опциональные поля
declared-stricter:             # §13.2.2, §10.10.2 — локальное ужесточение
  - clause: "QG-2"
    override: "required-negative-tc-per-clause"
    rationale: "regulated industry, double safety margin"
  - clause: "tc-pos-neg-pairing"
    override: "100% (без single-TC исключения для security)"
    rationale: "обязательное требование внутреннего ISMS"

exceptions: []                 # заявленные исключения относительно базового
уровня (с обоснованием в журнале аудита)

# Внешние записи conformance к §14.4 (опционально; поле значения см. ключ `claim`
ниже — носитель §14.6)
external-claims:
  - standard: "ISO/IEC 5338:2023"
    clause-ref: "§2.4.x"
    claim: "partial"           # full | partial | aligned
  - standard: "NIST AI RMF 1.0"
    clause-ref: "§2.4.x"
    claim: "aligned"

replaced-by: null              # нативный для носителя pointer на следующую
версию manifest, если выпущена
replaces: "CFM-2026-001@v2"    # нативный для носителя pointer на предыдущую
версию manifest
```

13.4.3 Семантика полей

- `renar-version` — версия стандарта; соответствие заявляется к конкретной точке стандарта; при выпуске minor-версии стандарта (§13.9) — обязателен re-assessment (§13.7) с обновлением `renar-version`.
- `senar-version` — версия SENAR, на которую опирается реализация; обязательна по MVR-7 (§0.5); её отсутствие в манифесте — несоответствие (§13.8).
- `manifest-id` — V1 immutable identifier; не переиспользуется даже после `replaced-by`.
- `level` — закрытый список §13.2; нарушение обязательные положения (§13.3) — соответствие отсутствует независимо от `level`.

- `level-target` — необязательное декларирование пути развития; не является нормативным обязательством.
- `assessment-mode` — `self` (§13.5) или `third-party` (§13.6); `third-party` обязан содержать формальную ссылку на внешний assessment-act.
- `assessor` — V6 author + role; для `third-party` — внешний участник с явной идентификацией.
- `next-assessment-due` — `assessment-date + cadence` (§13.7); просрочка — триггер потери соответствия (§13.8).
- `quality-gates` — обязательно содержит все пять gate-ids; статус `qg-0..qg-2` ∈ {required}; `qg-3, qg-4` ∈ {required, declared, absent}.
- `mandatory-clauses-confirmed` — каждое поле `true` обязательно для любого заявления о соответствии; `false` — манифест невалиден.
- `declared-stricter` — список локальных ужесточений; обязан содержать `clause`, `override`, `rationale`.
- `exceptions` — список заявленных исключений относительно базового уровня; каждое исключение требует обоснования в журнале аудита и не может затрагивать обязательные положения.
- `external-claims` — опциональный список записей соответствия внешним нормативным ссылкам (§14.4, §14.6); каждая запись содержит `standard`, `clause-ref`, поле **`claim`** (значение: full | partial | aligned) — технический ключ схемы, без замены термина в манифесте.

13.5 Процедура самооценки (Self-assessment)

13.5.1 Участник (actor)

Самооценка проводится **архитектором** проекта или **уполномоченным лицом** (глава 5), явно объявленным ответственным за соответствие.

13.5.2 Методология

Шаги: (1) контрольный список по §13.3 — участник проверяет каждое обязательное положение; результат в `mandatory-clauses-confirmed`; хотя бы одно `false` → оценка не завершается, формируется план устранения нарушений; (2) контрольный список по главе 11 §§11.4-11.8 для объявленного `level` — каждый критерий уровня = отдельная проверка с доказательной базой в носителе; (3) заполнение манифеста (все обязательные поля §13.4.2; `level` не выше пройденного контрольного списка); (4) подпись и публикация — участник подписывает манифест нативным механизмом (V6 author + timestamp); манифест становится частью источника истины через V3-рецензирование (для самооценки самоодобрение допустимо, доказательная база обязана фиксироваться).

13.5.3 Доказательная база (evidence)

Каждая проверка обязательного положения обязана сопровождаться ссылками на свидетельства в `audit-trail/CFM-<id>/<clause>.md` (V1 — стабильный идентификатор на конкретные артефакты, прогоны, события). Свидетельства хранятся бессрочно (V1 + §10.13.3 retention).

13.5.4 Периодичность

Первое заявление (kickoff); периодичность §13.7; триггер §13.8 (потеря соответствия); выпуск minor-версии стандарта (§13.9).

13.6 Оценка третьей стороной (Third-party assessment, опционально)

Опциональный путь подтверждения соответствия внешним оценщиком. Применяется в регуляторных контекстах (медицина, финтех, госсектор, AI-критические системы) или при контрактных обязательствах перед клиентом.

13.6.1 Участник (actor)

Внешний оценщик — независимый участник с read-only доступом к носителю на период оценки. Формальная квалификация специфична для носителя и фиксируется в `assessor.signature-ref` (внешний аудитор сертифицированной организации).

13.6.2 Методология

Та же что §13.5.2 (контрольный список §13.3 + контрольный список главы 11), плюс: журнал аудита — все действия оценщика (read-events) фиксируются нативно; независимая проверка — оценщик самостоятельно проверяет свидетельства без полагания на результаты самооценки; итог — подписанный манифест (оценщик подписывает нативным V6 механизмом) **или** отказ с обоснованием и списком конкретных нарушений.

13.6.3 Дополнительные поля манифеста

При `assessment-mode: third-party` обязательны:

```
third-party:
  assessor-organisation: "<наименование>"
  assessor-qualification-ref: "<pointer на квалификационный документ>"
  audit-report-ref: "<pointer на полный audit report>"
  audit-log-ref: "<pointer на read-events log>"
```

13.6.4 Соотношение self / third-party

Third-party **не отменяет** периодичность самооценки (§13.7); пути совместимы. Проект может вести самооценку ежеквартально и third-party ежегодно; манифест версионируется отдельно по каждому пути с разными `manifest-id`.

13.7 Период повторной оценки (Re-assessment cadence)

13.7.1 По умолчанию (Default)

Самооценка — **квартально** (каждые 3 месяца от `assessment-date`); third-party — **ежегодно**. Объявляется в поле `next-assessment-due` манифеста.

13.7.2 Переопределение (Override)

Периодичность может быть переопределена в `declared-stricter`:

```
declared-stricter:
  - clause: "re-assessment-cadence"
    override: "monthly"
    rationale: "AI-критический проект, eval-зависимость"
```

Ослабление периодичности (`override: "annual"` для self при default `quarterly`) **запрещено** — нарушение §13.3.1 инверсии источника истины (скрытое ослабление = сокрытие события потери соответствия).

13.7.3 Немедленные триггеры повторной оценки

Выпуск minor-версии стандарта RENAR (`renar-version` сдвигается); нарушение обязательное положение (§13.3, триггер потери соответствия §13.8); существенное изменение носителя (замена носителя — V1-V6 пересматривается); существенное изменение области применения (новый класс артефактов, новый SPEC-type).

13.8 Потеря соответствия (Loss of conformance)

13.8.1 Триггеры

Соответствие считается **потерянным** при наступлении любого из:

Trigger	Описание
Обязательное положение нарушено	Любая из §13.3.1–§13.3.7 перестала выполняться (например, появилась BR без ADAPT; носитель потерял V3 после миграции)
Деградация возможности носителя	V1, V2, V3, V4, V5 или V6 более не обеспечивается носителем (например, миграция на носитель без diff & review)
Манифест истёк	`next-assessment-due` прошёл без выпуска новой версии манифеста
Критерии уровня нарушены	Критерии заявленного уровня (см. главу 11) перестали выполняться без формального downgrade

Превышение порога метрики	Критическая метрика главы 12 превысила нормативный порог для уровня (например, Hallucination Rate > 5% на RENAR-4 — §12.3.3 явный триггер строки 94)
Внешний отказ	Внешний оценщик (third-party assessor) вернул отказ с обоснованием

13.8.2 Процедура

Шаги: (1) **регистрация события потери (loss-event)** в журнале аудита (`audit-trail/CFM-<id>/loss-events/<timestamp>.md`); (2) **формальный downgrade или декларация unknown-state** — downgrade (новая версия манифеста с `level` ниже текущего, если обязательные положения выполнены) либо unknown-state (явная декларация в публичных коммуникациях; манифест помечается `replaced-by: "<unknown-state>"` sentinel-значением, отличается от default `null` ; повторная оценка обязательна для восстановления); (3) **план восстановления** — `recovery/CFM-<id>-<timestamp>.md` с указанием срока и обязательные положения / уровневых критериев; (4) **повторная оценка** после плана восстановления — обязательно самооценка §13.5 с обновлённым манифестом; для third-party — повторная внешняя оценка.

13.8.3 Публичные коммуникации

Потеря соответствия, если уровень заявлен публично, обязана быть зарегистрирована публично в течение разумного срока (периодичность уведомления — `guide/`). Скрытие факта потери — нарушение §13.3.1 (журнал аудита источника истины).

13.9 Политика закрытого списка уровней RENAR-N

13.9.1 Нормативное правило

Закрытый список уровней RENAR-1..RENAR-5 (§13.2) **не расширяется** локально на уровне проекта. Любой проект, заявляющий соответствие, обязан указать `level` ∈ {RENAR-1, RENAR-2, RENAR-3, RENAR-4, RENAR-5}. Настоящая политика — специализация §1.7 политики закрытого списка для уровней зрелости; master-индекс — §1.7.5.

13.9.2 Что запрещено

Действие	Запрещено?	Почему
Локальное создание уровня на уровне проекта (`RENAR-6` , `RENAR-PRO`)	Запрещено	Нарушает закрытый список; соответствие не-портируемо
Локальное переопределение критериев (ослабление RENAR-4 без формального downgrade)	Запрещено	Нарушает контракт стандарта
Локальное ужесточение критериев (declare-stricter)	Разрешено	См. §13.4.2 `declared-stricter`
Заявление уровня выше фактически достигнутого	Запрещено	Нарушает §13.3.1 инверсия источника истины

Заявление о соответствии без манифеста	Запрещено	§13.4 обязательно
Промежуточные уровни типа `RENAR-3.5`	Запрещено	Список дискретный

13.9.3 Путь добавления нового уровня

Только через формальную процедуру изменения стандарта: исследовательский черновик (обоснование, типология критериев, сравнение с существующими RENAR-N) → публичное обсуждение → повышение minor-версии (`v1.X` или `v2.0`) → руководство по миграции (для существующих соответствующих проектов: изменения в чек-листе самооценки, новые поля манифеста).

Локальные для проекта расширения остаются за пределами соответствия — допустимы как internal-практики (`declared-stricter`), но **не** влияют на формальный `level` в манифесте.

13.10 Связь с другими главами

Глава	Связь
02 Положение в типологии методологий	§2.3 инверсия источника истины — обязательное положение §13.3.1; §2.7 следствие для соответствия явно отсылает к этой главе
06 Иерархия требований	`frontmatter` артефактов (BR / SR / TR) — критерии уровня RENAR-2/-3 проверяются по §6.5–§6.7 (глава 11)
07 ADAPT	Обязательное положение §13.3.3 (ADAPT для каждого T3); §7.4.4 закрытый список backward findings — §13.3.7
08 Specifications	Обязательное положение §13.3.4 (закрытый список 9 типов SPEC); §8.3.1 политика закрытого списка — §13.9
09 Test cases	Обязательное положение §13.3.5 (pos/neg парность); §9.10 QG-2 — §13.3.6
10 Жизненный цикл и QG	§10.3 канонические gates, §10.4.3 фрагмент манифеста соответствия — расширяется здесь до полной схемы манифеста; §10.10 политика закрытого списка для gates параллельна §13.9 для уровней
03 Версионирование носителя	Обязательное положение §13.3.2 (V1–V6 declared); §3.4 mapping таблица — не-нормативная, информационная
11 Maturity model	Детальные критерии уровней RENAR-1..5 — здесь §13.2 содержит семантическую сводку; полный контрольный список по уровню — в §§11.4–11.8 (по одной секции на уровень)
12 Metrics	Метрики, на которых строится самооценка (`approved-without-verified` , `pos-neg-pairing-percent` , и др.) — конкретизируются здесь как входные данные для §13.5

14. Нормативные ссылки

> ***Часть RENAR Standard v1.0-draft*** · *← Оглавление*

14.1 На какие стандарты опирается RENAR

У инженера, открывшего эту главу, один практический вопрос: какие из перечисленных стандартов мне реально соблюдать, а какие — просто фон? RENAR отвечает прямо. Чужие стандарты он не переписывает — опирается на них и заявляет соответствие лишь в части, а не целиком. Отсюда деление: **нормативные ссылки** (§14.4) — то, соответствие чему RENAR действительно декларирует (ISO/IEC/IEEE 29148, ISO/IEC 5338 и др.); **информативные ссылки** (§14.5) — методологии и терминология для позиционирования, для заявления о соответствии не обязательные; **позиция по соответствию** (§14.3) — одна формулировка, куда RENAR себя помещает среди родственных стандартов; **что RENAR не принимает** (§14.7) — закрытый список не заимствованных практик. Полный расширенный каталог информативных сопоставлений — reference/11.

При расхождении между нормативной ссылкой и положением этой главы побеждает положение этой главы (RENAR — специализация и адаптация). Заявление о соответствии внешнему стандарту RENAR делает **только** в указанной части, не целиком. Все даты — дата опубликования референсной редакции; ссылка является **датированной** (§14.4.1).

14.2 SENAR как родительский стандарт

RENAR — **специализация SENAR** (§2.1) в области инженерии требований. RENAR не дублирует следующие положения SENAR; они применяются как есть:

Положение SENAR	Используется без переписывания
5 ценностей и 14 правил	Контекст всего RENAR; см. §2.1
QG-0 / QG-1 / QG-2 как концепция	RENAR конкретизирует машины состояний в §10
10 общих метрик процесса	RENAR добавляет 10 доменных в §12.3
5 уровней общей зрелости	RENAR-M — отдельное измерение (§11.2)
5 ролей (Супервизор, AI-агент, Архитектор / Tech Lead, Рецензент, Заинтересованная сторона)	RENAR не переопределяет роли; см. §5
Инструментирование агентов (уровни контроля, профили)	RENAR расширяет для специфики требований

RENAR начинается там, где SENAR заканчивается: SENAR — общая методология AI-нативной разработки; RENAR — нормативный документ управления требованиями для SENAR-совместимых систем.

14.3 Позиция по соответствию

> ***RENAR — управление требованиями, согласованное с ISO/IEC/IEEE 29148:2018, адаптированное для разработки с AI-агентами, построенное поверх методологии SENAR и совместимое с координацией SAFe 6.0.***

Эта формулировка — точка отсчёта для всех заявлений о соответствии, которые проекты выпускают на основе RENAR (§13.4).

14.4 Нормативные датированные ссылки

Каждая запись §14.4.2–§14.4.6 содержит блоки **«Что нормирует»**, **«Как RENAR соотносится»** и **«Заявление о соответствии»**. Блоки **«Что RENAR адаптирует»** и **«Что RENAR не принимает»** есть только у записей глубокой адаптации (29148 и 5338); остальные записи короче — это отражает глубину заявления, а не дефект структуры.

14.4.1 Понятие датированной ссылки

RENAR использует **датированные ссылки**: каждая нормативная ссылка указывает конкретную редакцию стандарта (с годом). Ссылки без редакции не применяются — семантика меняется между изданиями, и заявление о соответствии должно быть проверяемо относительно зафиксированной редакции.

Жизненный цикл нормативной ссылки. *Активная* — ссылка указана в действующей версии RENAR (`renar-version` в манифесте соответствия, §13.4.2). *Обновлена* — референсный стандарт выпустил новую редакцию; RENAR обновляется в разумный срок (манифест проекта фиксирует `renar-version` , которая фиксирует редакции внешних ссылок). *Отозвана upstream* — референсный стандарт снят (как IEEE 830-1998); RENAR переносит ссылку в §14.5 и указывает правопреемника.

Триггеры немедленной переоценки. При выпуске новой редакции одной из ссылок §14.4 обязательна немедленная переоценка (§13.7.3) манифестов проектов: проверка глав RENAR на согласованность (запись в changelog); обновление `renar-version` в манифестах проектов; запись в audit-trail носителя.

Негативный сценарий: заявление о соответствии RENAR `renar-version: 1.0` при выходе новой редакции ISO/IEC 5338 **без** обновления `renar-version` в манифесте — невалидно; hook носителя (§13.8.1) обнаруживает устаревшую версию.

14.4.2 ISO/IEC/IEEE 29148:2018 (инженерия требований)

Официальное название (EN): Systems and software engineering — Life cycle processes — Requirements engineering.

Что нормирует: международный стандарт инженерии требований — потребности заинтересованных сторон, спецификация, валидация, верификация, атрибуты, трассируемость, жизненный цикл.

Как RENAR соотносится:

29148	RENAR	Тип

Классы требований: stakeholder, system, software	BR, SR, TR	Заимствует с переименованием
Атрибуты требования (18 в 29148)	Обязательный minimum во frontmatter (§6.5.2, §6.6.2) — 7–8 полей	Упрощает
Структура SRS	Структура носителя требований изоморфна	Заимствует
Методы верификации: inspection, analysis, demonstration, test	TC (§9) — полноценный артефакт; inspection — через workflow `[test-spec-change]` (§9.13)	Адаптирует

Что RENAR адаптирует:

- 29148 предусматривает 18 атрибутов; RENAR оставляет 7–8 обязательных, остальные — auto-derived (§4.12) или опциональны. Обоснование: при разработке с AI-агентами избыточная атрибутика увеличивает риск галлюцинаций (§12.3.3).
- 29148 не выделяет TC как отдельный артефакт. RENAR делает TC полноценным артефактом (§9).

Что RENAR не принимает: формальные обзорные совещания и прохождения из 29148 — заменены на QG-0 / QG-2 и состязательный AI-review (§11.7 для RENAR-4+).

Заявление о соответствии: RENAR заявляет соответствие ISO/IEC/IEEE 29148:2018 в части классов требований, атрибутов, жизненного цикла и методов верификации (фиксируется в манифесте, §13.4.2).

14.4.3 ISO/IEC 25010:2023 (модель качества продукта)

Официальное название (EN): SQuaRE — Product quality model.

Что нормирует: девять характеристик качества ПО (редакция 2023), включая новую Safety.

Как RENAR соотносится: характеристики 25010 — **обязательные категории** для нефункциональных SR. Frontmatter SR (§6.6.2) обязан содержать `quality-characteristic` из перечня 25010.

Заявление о соответствии: RENAR заявляет соответствие ISO/IEC 25010:2023 в части словаря категорий для NFR.

14.4.4 ISO/IEC 25022:2016 / 25023:2016 (меры качества)

Что нормирует: формальные меры для каждой характеристики 25010 (например, время отклика в мс).

Как RENAR соотносится: критерии Pass в TC (§9.11.1) обязаны быть выражены через меры 25022/25023, где применимо. Пример: «p95 < 200 мс при 100 RPS» вместо «производительность приемлемая».

Заявление о соответствии: RENAR заявляет соответствие в части измеримых Pass-критериев для TC.

14.4.5 ISO/IEC 5338:2023 (жизненный цикл AI-систем)

Что нормирует: первый международный стандарт жизненного цикла AI-систем (адаптация ISO/IEC 12207).

Как RENAR соотносится:

- **Журналы решений** — существенные решения AI-агента документируются; реализация: audit-trail (§10.13) + `ai-provenance` (§4.10.1).
- **Версионирование данных** — eval-datasets с version pin V5 (§3.3.5).
- **Версионирование модели** — `ai-provenance.generated-by` обязательно на RENAR-4+ (§11.7.1).
- **Непрерывная валидация** — eval-runs по расписанию (§11.8.1 для RENAR-5).

Что RENAR адаптирует:

- 5338 описывает полный жизненный цикл AI-системы (производный от ISO/IEC 12207); RENAR заимствует из него только процессы, касающиеся требований и происхождения AI-генерации артефактов — журналы решений, версионирование данных и модели, непрерывную валидацию — и выражает их через `ai-provenance` (§4.10.1) и уровни RENAR-4 / RENAR-5 (§11.7, §11.8).

Что RENAR не принимает: операционный слой жизненного цикла AI-систем — обучение, развёртывание и эксплуатацию моделей — он вне области RENAR (§1.3); стандарт нормирует только требовательную ось и происхождение AI-сгенерированных артефактов (`ai-provenance`).

Заявление о соответствии: RENAR заявляет соответствие в части жизненного цикла AI-артефактов и генерации артефактов с участием AI.

Негативный сценарий: заявление о соответствии 5338 без `ai-provenance` (§4.10.1) — невалидно. RENAR обязан отказывать в выпуске манифеста для таких проектов.

14.4.6 ISO/IEC 23894:2023 (управление рисками AI)

Что нормирует: классы AI-рисков и стратегии смягчения.

Как RENAR соотносится:

Риск (23894)	Смягчение в RENAR
Галлюцинации в выводе AI	Source citation (§4.10.2), метрика Hallucination Rate (§12.3.3)
Дрейф модели	pinning `last-run.requirement-version` (§9.12) + periodic re-run
Prompt injection через входные данные	Санитизация при импорте T3 (специфично для носителя, в guide/)
Bias в AI-генерации	Multi-model agreement для критических BR (RENAR-5, §11.8.1)
Состязательные входы	Состязательный обзор (§11.7.1, §11.8.3)
Single point of failure (одна модель)	Multi-model agreement; изоляция judge-модели (§9.13.4)

Заявление о соответствии: RENAR заявляет соответствие в части идентификации и смягчения AI-рисков в области требований; полный реестр — reference/03.

14.5 Информативные ссылки

Информативные ссылки — методологии и терминология для позиционирования; заявление о соответствии к ним RENAR **не** делает.

14.5.1 Пять ключевых (начните здесь)

Источник	Зачем RENAR
SAFe 6.0	Маппинг иерархии Epic/Feature/Story → BR/SR/TR (§4.13.1)
Spec-Driven Development	Инверсия источника истины (§2.3.1) как формальная парадигма
EARS (Mavin et al.)	Шаблоны формулировок SR и TC (§6.6.3)
BDD / Gherkin / Specification by Example	Prior art для полноценного TC (§9)
NIST AI RMF 1.0	Функциональное сопоставление Govern/Map/Measure/Manage

Краткие пояснения — в reference/11 §1–§2.

14.5.2 Расширенный каталог

Источник	Роль для RENAR
IEEE 830-1998 (deprecated)	Историческая ссылка; нормативный правопреемник — §14.4.2
BABOK v3	Терминология BA; gap по elicitation — в guide/
PMBOK 7	Принципы вместо процессов (§2.5)
ISTQB Foundation	Словарь тестирования, совместимый с `tc-type`
CMMI v2.0	Prior art для уровней зрелости (§11)
ISO/IEC 42001:2023	Организационный governance над RENAR
ISO/IEC 25059:2023	Качество AI-систем (расширение 25010)
EU AI Act (Reg. 2024/1689)	Поле `ai-act.risk-class` ; юридическое соответствие — вне RENAR
SysML / MBSE	Prior art «требования как граф» (reference/05)

Детальные сопоставления — reference/11.

14.6 Сводка соответствий

14.6.1 Сводная таблица

Стандарт	Тип	Уровень	Главы RENAR
SENAR	Родительский	Специализация	Все

ISO/IEC/IEEE 29148:2018	Нормативный	Высокий	06, 09
ISO/IEC 25010:2023	Нормативный	Средний	06, 08
ISO/IEC 25022/25023	Нормативный	Средний	09
ISO/IEC 5338:2023	Нормативный	Высокий	04 §4.10, 11
ISO/IEC 23894:2023	Нормативный	Средний	11, reference/03
Остальные (§14.5)	Информативный	—	см. reference/11

14.6.2 Совокупные заявления

манифест (§13.4.2) может содержать **несколько** заявлений о соответствии к ссылкам §14.4 одновременно. Каждое проверяется независимо. Частичное заявление не предусмотрено: проект либо соответствующий через RENAR, либо нет.

Обязательные положения и внешние стандарты. Обязательные положения §13.3 — внутренние требования RENAR. Заявления §14.4 — внешние, опциональные сверх минимума (например, RENAR-1 без заявления к 5338, если нет AI-генерации артефактов).

14.7 Что RENAR принципиально не принимает

Закрытый список практик из родственных стандартов:

Практика	Источник	Почему не принимается
Тяжёлые документные review meetings, IEEE 1028 inspections	RUP, SWEBOK	Несовместимо со скоростью AI-агентов; заменено состязательным AI-review (§11.7.1)
Только ручная верификация (inspection meetings 29148)	ISO/IEC/IEEE 29148 §6.4	Hooks носителя (§10.11) + AI-review
Process-first CMMI (CCB, OSP)	CMMI v2.0	Принципы + автоматическое обеспечение соблюдения (§2.5)
Формальные методы для всех требований (B, Z, TLA+)	Formal methods	Только critical safety domains, не базовый уровень
Недатированные ссылки на стандарты	Industry practice	Только датированные (§14.4.1)
Самодекларированное соответствие без манифеста	Отраслевая практика	манифест обязателен (§13.4)

14.8 Связь с другими главами

Глава	Связь
04 Terms	§4.13 — детальное сопоставление с родственными стандартами
02 Methodology	§2.3.4 SDD; §2.6 implications для обязательных положений

06 Hierarchy	frontmatter — реализация атрибутов 29148
09 Test cases	TC — расширение 29148; терминология ISTQB-совместима
10 Жизненный цикл и QG	QG — конкретизация SENAR QG-0..2
11 Maturity	Уровни RENAR-1..5
13 Соответствие	`renar-version`, `external-claims[]`
reference/11	Полный информативный каталог §14.5
reference/03 AI risk	Реестр рисков по 23894 + NIST

00. Быстрый старт

> *Сквозной пример: маленький проект «email/password sign-up». Полный цикл RENAR с минимальным набором артефактов: TZ → 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.*

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** §10.3 + **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 §27): git с PR-ревью; документо-ориентированное хранилище; БД с историей и подписями.

Структура папок (соглашение по умолчанию для git):

```
my-project.req/
  tz/                    # immutable TZ от клиента
  adapt/                 # ADAPT артефакты
  br/                    # Business Requirements
  sr/                    # System Requirements
  specs/{arch,api,data,ui,sec,ai,proc,int,ops}/
  tests/                 # TC (tc-type incl. contract)
```

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

Шаг 1. TZ (5 мин)

Клиент даёт небольшое TZ. 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"
---

# TZ-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-ref: "<ссылка>" }
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. ADAPT `approved` (5 мин)

Клиент даёт ответы. Lifecycle backward: `open → asked-to-client → answered → resolved → frozen (после approval)`.

Резюме после ответов:

```
### B-001: resolved (2026-05-18, Иванов И.И.)
«Показать сообщение + кнопку повторной отправки. 401 без объяснения — плохой UX.»
→ добавлен сценарий в Forward §2; создан SR-04.

### B-002: resolved (2026-05-18)
«Дата-центр в РФ, юрисдикция РФ. Provider выбирает исполнитель.»
→ data-residency: ["RU"] в Forward §2.

### B-003: resolved (2026-05-18)
«Повторная отправка не чаще 1 раза в 5 минут, не больше 5 раз в сутки.»
→ rate-limit правила в Forward §2; SR-04 уточнён.
```

Все backward → `resolved`, `open-questions-count = 0`. **QG-ADAPT-approve** — 6 пунктов passed:

```
status: approved
approval:
  client-signature: { signed-by: "Иванов И.И.", role: PM, organization:
"ClientCo", signed-at: "2026-05-18T15:30:00Z", signature-ref: "<ссылка>" }
  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.

Шаг 4. BR-01 (3 мин)

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

```
---
id: BR-01
title: "Регистрация пользователей через email"
type: BR
status: approved
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
status: approved
parent: { id: BR-01 }
source: { adapt: "ADAPT-001", adapt-section: "Forward §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
status: approved
source: { adapt: "ADAPT-001" }
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
status: ready
verifies: [{ id: SR-01, requirement-version: "1.0" }]
negative: false
automation: { status: automated, 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
status: ready
verifies: [{ id: SR-01, requirement-version: "1.0" }]
negative: true
automation: { status: automated, 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. Запустить TC и promote SR (2 мин)

Test runner на носителе запускает TC 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>", requirement-version: "1.0" }
```

Когда **все** TC из `SR-01.verified-by[]` зелёные → выборочная проверка (Правило 5 Core: инженер вручную запускает 1-2 случайных passing TC и сверяет с SR). Если spot-check пройден → **QG-2 Verification Gate** пройден → SR-01 → `verified`:

```
# sr/SR-01-sign-up.md:
status: verified
verified-by: ["TC-01", "TC-02"]
verified-at: "2026-05-19T14:00:00Z"
verified-by-engineer: "Петров П.П."
```

Аналогично — SR-02, SR-03, SR-04. Когда все SR в BR-01 verified → QG-4 (BR ready for acceptance).

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)
       └─ constrained-by SPEC-API-01 (REST contract)
                                  └─ depends-on SPEC-DATA-01, SPEC-SEC-01
```

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 (T3).
- Прошли двустороннюю интерпретацию через ADAPT с тремя backward findings → клиент дал ответы → approved.

- 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 два шлюза качества: QG-ADAPT-approve (≡ QG-3 Architecture) и QG-2 Verification Gate.

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 T3 и delta-T3 с adversarial reviewer — в 01-walkthrough.md фаза 2 + фаза 8.

Что дальше

Хотите...	Документ
Детальный сквозной пример (login + 2FA + 9 фаз)	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 (15 глав)	`standard/`
Схемы артефактов и validation rules	`reference/02-schemas.md`
Глоссарий и mapping на отраслевые стандарты	`reference/01-glossary.md`

Быстрый старт RENAR 1.0-draft — renar.tech

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.*

Маршрут читателя. Фазы 0–2 — сбор контекста, подписание ТЗ, ADAPT. 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 приёмочный контур QG-4.

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 (двусторонняя 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 resolved или reclassified; Forward охватывает §2 + §3 ТЗ полностью.

2.4 ADAPT в статусе `approved`:

```yaml
---
id: ADAPT-001
title: "Адаптация TZ-2026-042 — Login Flow AcmeCorp"
type: ADAPT
source-tz: { id: TZ-2026-042, signed-date: "2026-05-03", signed-by-client:
"AcmeCorp PM" }
status: approved
approval:
  client-signature: { signed-by: "Иванова А.А.", role: "Product Lead",
organization: "AcmeCorp", signed-at: "2026-05-04T11:30:00Z" }
  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**.

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
status: approved
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 §14.1 Forward; 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 §14.1 Forward).

## 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
status: approved
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 (Правило 4 RENAR Core: для каждого assertion в SR — 1 pos + 1 neg TC).

TC-001 (позитивный):

```
---
id: TC-001
title: "Регистрация с разрешённым доменом — happy path"
type: TC
tc-type: system
status: ready
verifies: [{ id: SR-01, requirement-version: "1.0" }]
negative: false
automation: { status: automated, location:
"tests/auth/test_registration.py::test_allowed_domain_succeeds", runner: pytest }
---

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

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

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

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

```
## Not in 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
status: ready
verifies: [{ id: SR-01, requirement-version: "1.0" }]
negative: true
automation: { status: automated, location:
"tests/auth/test_registration.py::test_disallowed_domain_rejected", runner:
pytest }
---

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

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

## Then (Pass)
- 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 criteria
- status ≠ 422; User создан; email отправлен (security leak); audit-запись
отсутствует.
```

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 BR-01 (`draft → approved`). Предусловия: `source.adapt = ADAPT-001 (approved)` ; дерево SR в допустимом состоянии перед утверждением BR; adversarial-review успешен; утверждения и связи cite разделы ADAPT-001. Постусловие: BR-01 + дочерние SR при необходимости каскад `draft → approved` .

5.2 QG-1 — только TC: `draft → ready` . По §10.3.2 QG-1 применим только к TC и отделяет подготовленный контрольный пример с исполнимой реализацией проверки от черновика. Предусловия для TC: зафиксирован version-pin реализации (V5); `automation.status` + `location` валидны; статические проверки пройдены; pos/neg парность (§9.7); обязательные секции body заполнены. Постусловие: `draft → ready` .

После **QG-0** на BR/SR и **QG-1** на каждом TC можно открывать работу по TR (фаза 6).

Фаза 6 — Реализация

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`, `requirement-version = 1.0` в TC файлах.

7.2 Выборочная проверка (Правило 5 Core). Раз в спринт инженер вручную запускает 5 случайных passing TC и сверяет фактический результат с SR. Selected: TC-001, TC-008, TC-012, TC-019, TC-024 → 5/5 совпадают.

7.3 Promote SR-01 → verified. QG-2 предусловия: approved ADAPT linkage; pos/neg TC passing; `last-run.requirement-version` зафиксирован; выборочная проверка пройдена. Постусловие: SR-01 `approved → verified`; 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. 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). После 1 итерации с клиентом → approved.

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

```
Affected:
  BR-01 (расширение охвата)
  SR-01: verified → approved (TC rerun required)
  TC-001..004: re-pin 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. После spot-check — SR-01 v1.1 → verified снова.

> ***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`*. Снимает overhead двойной подписи для тривиальных изменений (e.g., переименование поля).*

Фаза 9 — QG-4 (приёмка)

9.1 Через 4 недели после release. Window: 4 weeks post-release.

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

9.2 QG-4 report + adversarial. AI генерирует acceptance report. Adversarial critic находит: «recovery через admin не покрыт TC, верифицирующим full flow с tickets». Архитектор соглашается → создаёт mini-delta для добавления acceptance TC.

9.3 Sign-off. После закрытия находок клиент подписывает acceptance. BR → status `accepted`. Архив отчёта: `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)
├── 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)
├── tz/                   TZ-2026-042.md + TZ-2026-051.md (delta, immutable)
├── elicitation/          # артефакты фазы 0
├── lessons/2026-Q2.md    # уроки фазы 9
└── QG-4-REPORT-v1.0.md   # acceptance report
```

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%
Acceptance disputes	0 (1 finding, resolved before sign-off)
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 T3 до passing TC за несколько операций запроса к носителю.
4. **Дельта-Т3** — затронутые SR/SPEC/TC/TR вычисляются автоматически.
5. **Замыкание контура** — QG-4 связывает результат с бизнес-метриками (KPI achievement).
6. **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 1.0-draft — 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; для каждого ТЗ есть ADAPT.

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. Для каждого ТЗ создаётся `ADAPT` — артефакт-мост: раздел Forward («как мы поняли») и backward (вопросы заказчику).
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 первом же спорном acceptance.
- **«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/`, `dpia/`, ...
3. Изменения требований — только через delta-T3 (новый неизменяемый артефакт), не через прямое редактирование.
4. T3 зафиксировано как неизменяемое (изменения только через delta-T3).

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"
status: approved          # 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. Lifecycle статусы используются реально: каждый артефакт в одном из закрытых состояний (`draft` / `approved` / `verified` / `deprecated` / `obsolete`).

3. TC создаются для всех priority=must BR / SR / SPEC.
4. Нативный для носителя COVERAGE report auto-generated на каждый promote-transition.
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 утверждённых артефактов есть успешно пройденные контрольные примеры (`TC`); переход 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% approved артефактов имеют `verified-by` ссылку на ≥ 1 TC.
2. Pos/neg парность для каждого нормативного утверждения.
3. Контрольная точка `QG-2` (`Verification Gate`) блокируется встроенными в носитель проверками: перевод в `verified` только при всех успешных `TC` для текущей `requirement-version`.
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: минимум `generated-by` + `generated-at`.
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.

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-ТЗ.
3. Через 1-2 итерации требование «созревает» — становится full-RENAR (без legacy маркера, с полным frontmatter и ТС).

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 ТС 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-T3. 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 Core (5 правил + ADAPT) — порядка полдня-дня на инженера.
- Дисциплина ADAPT на каждое T3 — дополнительные часы на elicitation / backward до старта кода (окупаются отсутствием переоткрытия T3 позднее).
- Носитель уже есть (git) — отдельных лицензий не требуется; tooling-автоматизация опциональна и вводится постепенно.

Выгода (что возвращается):

- Сокращение времени декомпозиции T3 с AI-ускорением (порядок величин — standard/12 §12.5.1: 5–10× на RENAR-4, иллюстративно).
- Меньше споров на приёмке: ADAPT с двойной подписью фиксирует интерпретацию до кода.
- Меньше инцидентов из негативных сценариев — за счёт обязательной pos/neg-парности TC.
- 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)
│   ├── adapt/                    ← ADAPT-NN.md (bridge artefacts)
│   ├── 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/
│   ├── tr/                       ← TR-NN.md (task requirements)
│   ├── tests/                    ← TC-NN.md (контрольные примеры, `TC` —
самостоятельные артефакты)
│   ├── 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	Feature-ветки (`draft` / `review`) против `main` (`approved`); PR — набор изменений
V5 — сквозная фиксация версии	Реализация ссылается на конкретную версию требования	Submodule-pin между `.req` и `.src`; commit SHA / `requirement-version` в `verifies[]`
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-ТЗ:** 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 актуальные SR / SPEC через `verifies[].version`.
- Новые / изменённые TC соответствуют контракту в `.req`.
- Никаких изменений Pass/Fail-критериев TC без `[test-spec-change]` тега.

Утверждение = QG-2 (standard/10 §10.3.3) — после прохождения автоматизированных TC.

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 versions pinned to current requirement-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. T3 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. **Создание ADAPT:** `adapt/ADAPT-001.md` — Forward (интерпретация по каждому § ТЗ) + Backward (вопросы клиенту), standard/07. Backward отрабатывается с клиентом до approval.
4. **Двойная подпись ADAPT → QG-ADAPT-approve:** ADAPT в `approved` (клиент + архитектор), `open-questions-count == 0`.
5. **Декомпозиция:** AI-агент выводит BR из approved ADAPT, затем SR (`source.adapt`), SPEC (`constrained-by[]`) и парные pos/neg TC.
6. **QG-0 approval** каждого артефакта (`draft → approved`); CI dry-run новых TC.
7. **PR в `.req` → merge;** bot regenerates REQUIREMENTS.md / COVERAGE.md / TEST-PLAN.md.
8. **Setup `.src`:** добавить submodule `requirements/ → <project>.req @ <commit>`, создать TR, реализовать, QG-2.

7.2 Delta-ТЗ workflow

Полная последовательность для применения нового delta-ТЗ.

1. **branch в `.req`:** `git checkout -b change/TZ-2026-042` в `<project>.req`.
2. **Создание delta-ТЗ + delta-ADAPT:** `tz/TZ-2026-042-delta.md` (текст delta) и `adapt/ADAPT-NNN-delta.md` (forward интерпретация + backward findings, standard/07 §7.6). Delta-ADAPT обязан пройти двойную подпись прежде чем затронутые требования будут модифицированы.
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:** version++, status: `approved`, 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 — утверждение → AI-агент переводит требование в `verified`.

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-draft:

- ✅ **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 + 9 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 → ADAPT → BR) сейчас не задокументировано как отдельный workflow рядом с §7 (delta-workflow); шаг 2 в §7 явно использует delta-ADAPT согласно standard/07 §7.6. Утверждённый forward-workflow — задача отдельного draft главы.
- ⏳ **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" tc/`
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 requirement-version SR-12	`yq '.version' sr/SR-12.md`

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)

> ***Informative appendix.*** *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**) (§3.2–11.3).

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 (§3.5).

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 (5 правил, ADAPT, 2 QG) 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 в группе со статусом `verified`
Capability	BR подсистемы (если у подсистемы свой stakeholder)	`<subsystem>.req/br/`	Solution Architect	Все BR подсистемы `verified`
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 на текущей `requirement-version` (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	TR с `level: research` + Decision в decision log	`<subsystem>.req/tr/` + decision log	Команда	Decision записан и связан с originating SR
Defect	TR + ссылка на нарушенный SR через `defect-of`	task tracker + `linked_defects` в SR	Команда	Negative TC проходит, regression test добавлен

Запрещённые соответствия: INT-SR — устаревший термин (§4.3 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"
status: approved
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`.

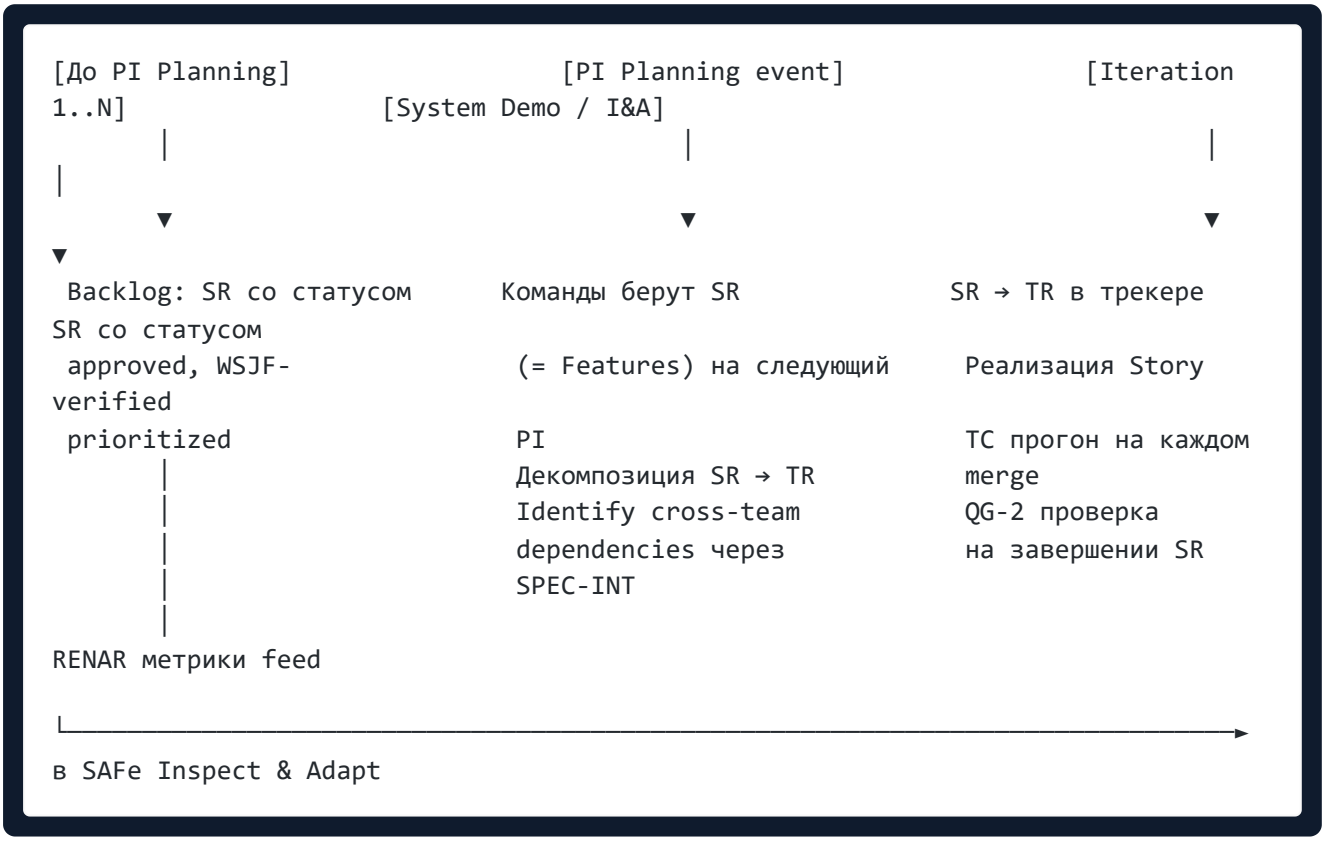
---

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 со статусом `proposed` или `approved`	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 `in-progress`
System Demo	SR со статусом `verified`	Evidence из 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 (готовность к старту) и QG-2 (verified)
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"
status: approved
constrained-by:
```

```
  - id: SPEC-INT-01
    ref: "specs/int/SPEC-INT-01-auth-handshake.md"
    requirement-version: "1.2"
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"
status: approved
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 в группе → `verified`, KPI impact подтверждён через outcome metric
Capability	BR подсистемы	Все BR со статусом `verified`; outcome metric измерен
Feature	SR	QG-2 passed: все TC из `verified-by` имеют `last-run.result = pass` на текущей `requirement-version`
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 создаются **до** реализации; QG-0 требует `verified-by` пустым только при `status: proposed`, не `approved`
Refactoring	Continuous reconciliation hook (§7.5 ADAPT); 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 T3	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 (SR verified)	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 ТЗ 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»	**Drift Lag** (reconciliation)

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 9 типов SPEC, §4.4).

11.3 Добавить (новое в RENAR)

- **ADAPT** — bridge artefact между T3 и иерархией требований. В SAFe нет прямого аналога; реализуется как обязательный этап перед декомпозицией в BR.
- **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 + 9 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 9 типов SPEC, включая SPEC-INT.

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`; `requirement-version`. 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)

Audit deliverable. 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

Data residency enforcement. 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 + 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	Acceptance TC + spot-check
Operation	Reconciliation hook (drift detection, §4.11)
Maintenance	Delta-T3 + 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

CHD scope minimization. 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 есть ровно один approved ADAPT с двойной подписью.
- Все 9 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 §14).
- `assessment-date` свежее, `next-assessment-due` не просрочен.

Все 7 — `yes` → проект conformant к заявленному `level`. Хотя бы один `no` → manifest не выпускается (§13.5.2).

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

```
renar-version: "1.0"
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"]
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 §14.

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-residency.region: eu-west`), 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: Статусы (`proposed` / `approved` / `verified` / `obsolete`) 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 существуют и `verified` .

Предотвращение: ID требований **неизменяемы** — переименование запрещено. Deprecated требования остаются в репозитории со статусом `obsolete` , 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; переприменить в правильном порядке.

2.7 TC ↔ requirement provenance drift

Симптом: TC верифицирует требование, но требование уже изменилось — `last-run.requirement-version` ниже текущей `version` требования. Тест зелёный, но проверяет устаревшее поведение.

Обнаружение: Coverage report показывает категорию «Stale» — TC с устаревшей `last-run.requirement-version`. Reconciliation ловит это автоматически (по V5-pin версии).

Предотвращение: В TC обязательное поле `verifies[].requirement-version` — pinned версия. QG-2 запрещает перевод требования в `verified`, если хотя бы один TC из `verified-by` имеет stale `last-run`.

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 T3 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)

> ***Informative.*** *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	Каждый `approved` TC в scope имеет `tc-type` и `verified-by[]`
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. §5.6)
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 — §4.5 (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 ADAPT как формальность

Симптом: Клиент / stakeholder не вычитывает ADAPT перед подписанием. Backward-секция (вопросы клиенту) пустая или содержит yes/no без контекста.

Признак: ADAPT approved за < 24 часов после генерации; rate of disputed requirements at acceptance растёт.

Смягчение: Двойная подпись ADAPT (standard/05-roles §5.5) — обязательны и стейкхолдер, и архитектор. Backward-секция обязана содержать ≥ 1 не риторический вопрос. Spot-check 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). TC без `last-run` за N дней автоматически помечаются `stale`; QG-2 блокирует, пока не перепрогнаны.

---

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 → repump TC на текущей requirement-version.
- Test-fitting → revert критерии; root cause AI-агента.
- Organizational → process retro + специфичные mitigations (§5).

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 §2 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 (§5).** Сейчас даны качественные «признаки». Набор допустимых значений понадобится после накопления полевых данных. Ответственные: команда стандарта RENAR и внедренческие организации.
- **Формальный замер коэффициента «техавтобусности» для §5.7.** 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.

---

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 §14: 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; добавляете
```

```
запись в source[]
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 §14.4; 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 Supervisor/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) → создать ветку `feat/TASK-NN-slug` в `.src` → реализовать + написать тесты по AC → MR/PR → ревью → merge.

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 Supervisor/Architect → 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-index.md` (список изменённых требований) → проверить, затрагивают ли изменения ваши текущие задачи → если да — обновить или закрыть по согласованию с Supervisor.

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` → T3
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 T3?	`acme-platform.req/tz/TZ-YYYY-NNN-index.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-T3 — дополнение к T3 при повторном заказе; deprecated — устаревшее требование (не удаляется, только помечается).

09. Библиотека примеров

> *Шесть сценариев (E1–E6): три полных цикла in-doc (E3–E6) + два walkthrough (E1–E2). Каждый in-doc пример: **T3** → **ADAPT** → **BR/SR** → **SPEC** → **TC** → **QG**.*

#	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)	§14	Legal, PM, Architect	Compliance, SPEC-DATA/SEC
E4	Webhook idempotency (REST API)	§4	Backend, Architect	`tc-type: contract`, SPEC-API
E5	RAG-ассистент (SPEC-AI eval)	§5	AI engineer	`tc-type: eval`, judge isolation
E6	Delta-T3: scope dispute	§5	PM, Client, Architect	ADAPT backward, неизменяемое T3

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"
status: approved
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"
status: approved
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"
status: approved
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
status: ready
verifies:
  - id: SR-08
    requirement-version: "1.0"
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
status: ready
verifies:
  - id: SR-08
    requirement-version: "1.0"
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 `requirement-version` (`1.0`)

2.7 Что дальше

- Полный сценарий под `git` — 03-tool-guide-git
- Compliance mapping — 06-compliance
- Conformance manifest — reference/08

---

3. E4 — Webhook idempotency (SPEC-API)

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

ТЗ (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, requirement-version: "1.0" }]
negative: false
```

```yaml
id: TC-201
type: TC
tc-type: contract
verifies: [{ id: SR-20, requirement-version: "1.0" }]
negative: true
```

Neg: второй key с другим body → 409 Conflict.

Контрольная точка `QG-2`: оба `TC` успешны; в `last-run` совпадает `requirement-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, requirement-version: "1.0" }]
automation:
  judge-model: "eval-judge-v2"   # ≠ production chat model
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 §4.5.

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** → новый ADAPT-002b (forward + backward).
3. Backward finding: «PDF export не входил в Forward §3 ADAPT-002».
4. Клиент подписывает delta-ADAPT → новые SR/SPEC/TC.

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

status: approved
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"
```

```yaml
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-NOTIFY-API-01, SPEC-NOTIFY-OPS-01
```

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 — в статусе `approved` (или `verified`).
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.

---

10. Миграция на v1.0-draft

> *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 ТЗ + 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-draft)

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

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 §14 (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: `source.adapt` на approved ADAPT.

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 §14.

Фаза D — manifest (1 день)

1. Bump `renar-version: "1.0-draft"`.
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 ADAPT «только для новых T3»	Non-conformant для legacy TZ	Ретроспективный ADAPT на каждое активное T3

---

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 1.0-draft — renar.tech

Глоссарий RENAR

> ***Назначение:*** *единый источник канонических терминов RENAR с примерами и mapping на отраслевые стандарты. При расхождении формулировок* ***нормативно*** *побеждает* `standard/04-terms.md` *; этот глоссарий — informative lookup для читателя и assessor-а.*

1. Цепочка авторитетности

В случае разногласий по термину применяется порядок:

1. `standard/04-terms.md` — нормативный канон: определения главы стандарта побеждают при любом расхождении.
2. **Этот документ** — информационные разъяснения и сопоставление с отраслевыми стандартами; не переопределяет `standard/04` .
3. **ISO/IEC/IEEE 29148** — международный стандарт инженерии требований.
4. **BABOK Guide v3** — свод знаний по бизнес-анализу (*Business Analysis Body of Knowledge*).
5. **Изменение через предложение поправки в полный стандарт RENAR** — если все источники молчат.

Закрытые списки: главный индекс шестнадцати закрытых списков — `standard/01 §1.7.5` .

Не используются как источник терминов: тикеты в багтрекере, чаты команды (slang), устаревшие презентации, маркетинговые материалы.

2. Канонические термины

2.1 Уровни требований

Каноника v1.0 — закрытый список (см. `standard/04-terms.md §4.3`):

RENAR (канонический)	Полное название	RU (для UI)	Описание
BR	Business Requirement	Бизнес-требование	Бизнес-цель уровня заказчика: что организация хочет получить.
SR	System Requirement	Системное требование	Инженерное требование к ПО; поддаётся проверке. Один `SR` — одна проверяемая единица.
TR	Task Requirement	Требование к задаче	Требование уровня реализации, выводимое из `SR` ; единица планирования работ.

TC	Test Case	Контрольный пример (TC)	Проверяемый артефакт, подтверждающий `SR` / `BR`. Описывает поведение, а не реализацию.

Правило идентификации: в `frontmatter` `id` — канонический идентификатор RENAR (`BR-01`, `SR-05`). При экспорте в другой носитель применяется целевое сопоставление.

Уточнения для `SR` (по полям frontmatter, не отдельные типы артефактов):

- **Module SR** — `SR` с `level: module` (`standard/06 §6.7`). Раньше отдельный label `TM` (§2.1.1).
- **Integration SR** — `SR` с `constrained-by: [SPEC-INT-N]`. Раньше отдельный label `INT-SR` (§2.1.1).
- **Contract TC** — `TC` с `tc-type: contract`. Раньше отдельный label `INT-TC` (§2.1.1).

Семейство `SPEC` (UX / ИИ / архитектура / интеграционные спецификации) — см. §14.5 Семейство `SPEC`.

2.1.1 Legacy labels (deprecated)

При миграции с материалов до v1.0 draft встречаются устаревшие метки. Их замены в канонике v1.0 (`standard/04-terms.md §4.14.1`):

Устаревший label	Замена в канонике v1.0	Пояснение
`TM` (Module/Submodule SR)	`SR` с `level: module`	Уточнение `SR`, не отдельный тип артефакта
`UIC` (UI Concept)	`SPEC-UI` (`standard/08 §8.5.6`)	Часть семейства `SPEC` (§14.5)
`AIC` (AI Concept)	`SPEC-AI` (`standard/08 §8.5.7`)	Часть семейства `SPEC`
`INT-SR` (Integration SR)	`SR` с `constrained-by: [SPEC-INT-N]`	Подкласс `SR`
`INT-TC` (Integration TC)	`TC` с `tc-type: contract`	Подкласс `TC`
`TS` (Technical Specification)	`SPEC-ARCH` или `SPEC-OPS` в зависимости от содержания	Часть семейства `SPEC`

Миграция существующих артефактов: встроенная в носитель привязка к набору изменений должна автоматически распознавать устаревшие метки и предлагать канонические замены. Каноническая таблица сопоставления legacy → v1.0 — `standard/04-terms.md §4.14.1`.

2.2 ADAPT artifact family

Термин	Описание
ADAPT	Адаптивный документ, формулирующий проектное ТЗ (*Adaptive Document for Articulating Project's TZ*). Двусторонняя инженерная адаптация ТЗ: прямое направление (интерпретация) и обратное (вопросы). Утверждается двойной подписью.

delta-ADAPT	`ADAPT` для delta-ТЗ. Цепочка: ADAPT-001 → ADAPT-001-delta-1 → ADAPT-001-delta-2; применяются по порядку.
errata-ADAPT	Правка утверждённого `ADAPT` (наша ошибка интерпретации). Не меняет замороженный документ — добавляется отдельный артефакт.
Forward (в ADAPT)	Инженерная интерпретация каждого раздела ТЗ: цитата → интерпретация → достроенные сценарии → охват.
Backward (в ADAPT)	Список проблем и вопросов к клиенту. Жизненный цикл: `open` → `asked-to-client` → `answered` → `resolved` → `frozen`.
Term mapping (в ADAPT)	Таблица «термин клиента → инженерное понимание».

2.3 Backward categories (закрытый список)

ADAPT backward записи относятся к одной из 7 категорий. Список закрыт; новые добавляются только через изменение полного RENAR Standard:

Категория	Описание
`contradiction`	Противоречие внутри ТЗ.
`gap`	Гэп — про это ТЗ молчит.
`hidden-assumption`	Скрытое предположение инженера, требующее подтверждения.
`feasibility`	Технически нереализуемое или дорогое требование.
`regulatory`	Затрагивает законодательство / compliance.
`terminology`	Неясный или конфликтующий термин клиента.
`scope`	Уточнение границ scope (входит / не входит).

2.4 Контрольные точки качества (`Quality Gates` , закрытый список, каноника v1.0)

Закрытый список контрольных точек RENAR (по `standard/10-lifecycle-qg.md §10.3–10.4`):

Gate	Применяется к	Условие пропуска
QG-0 Контрольная точка утверждения (`Approval Gate`)	Перевод `BR` / `SR` / `SPEC` : `draft` → `approved`	Цель, критерии приёмки, не менее одного негативного сценария и охват; для `SR` — родительский `BR` в `approved` +; для `SR` с `constrained-by` — `SPEC` в `approved` +.
QG-1 Контрольная точка реализации (`Implementation Gate`)	Перевод `TC` : `draft` → `ready` (**только** для `TC`)	У `TC` поле `verifies` ссылается на артефакт в `approved` +; `requirement-version` совпадает; охват реализации и закрепление версии зафиксированы.

QG-2 Контрольная точка проверки (`Verification Gate`)	Перевод `SR` / `BR` : `approved` → `verified`	Все `TC` из `verified-by` зелёные; хотя бы один `TC` с `negative: true` ; у `last-run` совпадает `requirement-version` с текущей `version` ; выборочная проверка пройдена.
QG-3 Контрольная точка архитектуры (`Architecture Gate` , *опционально*)	Утверждение `SPEC-ARCH` / двойная подпись	Зафиксирован артефакт в духе ADR в носителе; двойная подпись (клиент + архитектор). Не обязательна для декларируемого соответствия RENAR.
QG-4 Контрольная точка приёмки (`Acceptance Gate` , *опционально*)	Перевод `BR` : `verified` → `accepted`	Все `BR` покрыты проверенными `SR` ; приёмочные `TC` (`tc-type: acceptance`) зелёные; подпись клиента. Не обязательна для декларируемого соответствия RENAR.

Соответствие стандарту RENAR (соответствие): `QG-0` / `QG-1` / `QG-2` обязательны для декларируемого соответствия (`standard/13 §13.3`). `QG-3` / `QG-4` — необязательные расширения.

Список закрыт; новые номера контрольных точек добавляются только через формальную процедуру изменения полного стандарта RENAR.

2.4.1 Устаревшие имена QG (deprecated)

До v1.0 в RENAR использовались иные имена контрольных точек. Сопоставление по `standard/04-terms.md §4.14.1` :

Устаревшее (до v1.0)	Замена в канонике v1.0	Пояснение
`QG-0 Context Gate`	`QG-0 Approval Gate`	Только переименование; смысл сохранён
`QG-1 Requirements Gate`	`QG-1 Implementation Gate`	**Сдвиг смысла:** ранее утверждение `BR` / `SR` , теперь только перевод `TC` : `draft` → `ready`
`QG-2 Implementation Gate`	`QG-1 Implementation Gate`	Перенумерация и слияние в одну `QG-1`
`QG-3 Verification Gate`	`QG-2 Verification Gate`	Перенумерация
`QG-4 Acceptance Gate`	`QG-4 Acceptance Gate`	Имя то же; теперь **опционально** для декларируемого соответствия RENAR

Миграция существующих артефактов: средства автоматизации преобразуют ссылки по таблице выше. Каноническая таблица сопоставления legacy QG → v1.0 — `standard/04-terms.md §4.14.1` .

2.4.2 Локальные контрольные точки ADAPT

Жизненный цикл `ADAPT` (`standard/07-adapt.md`) использует локальные контрольные точки `ADAPT` параллельно каноническим `QG-N` . Это не отдельная нумерация `QG` , а локальные события жизненного цикла артефакта `ADAPT` :

Контрольная точка	Применение	Условие прохода
QG-ADAPT-draft	Создание `ADAPT`	Раздел Forward охватывает все разделы T3.
QG-ADAPT-review	Перевод в `review`	Все записи backward — `open` или `asked-to-client`; нет состояния `draft`.
QG-ADAPT-client-ready	Передача клиенту	Все backward — `asked-to-client`; пакет вопросов собран.
QG-ADAPT-answered	После ответа клиента	Все backward — `answered`.
QG-ADAPT-approve	Утверждение `ADAPT`	Все backward — `resolved`; двойная подпись (клиент + архитектор).
QG-ADAPT-frozen	После утверждения	Неизменяемость; разрешена генерация `BR` / `SR` / `SPEC`.

Локальные контрольные точки `ADAPT` **не входят** в набор `QG-0` / `QG-1` / `QG-2` для декларируемого соответствия RENAR. Реализация может выразить `QG-ADAPT-approve` как локальный псевдоним для `QG-3` (`Architecture Gate` , если применимо) либо хранить отдельно — на усмотрение носителя.

2.5 Семейство SPEC (закрытый список)

Тип	Назначение	Источник
SPEC-ARCH	Архитектура системы / подсистемы: контексты, контейнеры, компоненты, представление развёртывания, атрибуты качества	Раздел Forward `ADAPT`
SPEC-API	Контракты API (REST / GraphQL / gRPC / асинхронные события); версионирование, модель ошибок, ограничения частоты	Раздел Forward `ADAPT`
SPEC-DATA	Модель данных: схема, ER-диаграмма, индексы, миграции, хранение, классификация персональных данных	Раздел Forward `ADAPT`
SPEC-INT	Интеграция: взаимодействие между подсистемами и внешними системами; протоколы, контракты, SLA	Раздел Forward `ADAPT`
SPEC-PROC	Процесс / рабочий поток: бизнес-процессы, машины состояний, паттерны saga, оркестровка и хореография	Раздел Forward `ADAPT`
SPEC-UI	UI / UX: экраны, навигация, пользовательские сценарии, доступность, локализация, эталонные изображения	Раздел Forward `ADAPT`
SPEC-AI	ИИ / ML: карточки моделей, RAG, инженерия промптов, стратегия оценки, бюджет стоимости	Раздел Forward `ADAPT`
SPEC-SEC	Безопасность: аутентификация / авторизация, модель угроз, управление секретами, классификация данных	Раздел Forward `ADAPT`, регуляторные замечания backward

SPEC-OPS	Эксплуатация: развёртывание, наблюдаемость, SLO / SLA, runbook, аварийное восстановление	Раздел Forward `ADAPT`

Список закрыт — ровно девять типов (каноника по `standard/08 §8.3`); новые типы SPEC добавляются только через изменение полного стандарта RENAR.

2.6 Статусы жизненного цикла

Статус	Применим к	Значение
`draft`	все артефакты	В работе, не для использования другими
`review`	все	На ревью, возможны изменения
`approved`	ADAPT, SR, SPEC	Утверждён; неизменяем после двойной подписи (для `ADAPT`) или одной (для `SR` / `SPEC`)
`verified`	SR	Подтверждён успешными `TC` и выборочной проверкой
`frozen`	ADAPT	Утверждён и неизменяем; используется как источник для генерации `BR` / `SR` / `SPEC`
`deprecated`	все	Устарел, не используется в новых артефактах; замена (если есть) указывается полем `replaced-by`
`obsolete`	research/legacy	Полностью изъят из обращения

2.7 Возможности носителя (V1–V6)

RENAR не привязан к конкретному носителю артефактов. Артефакт может находиться в любом носителе, который удовлетворяет следующим возможностям:

Нормативное определение — `standard/03 §3.3`:

Возможность	Описание
V1 — неизменяемая история	Любое прошлое состояние артефакта можно адресно восстановить без потерь (цепочка ревизий для аудита сохраняется).
V2 — атомарная единица изменения	Изменение артефакта (или согласованной группы) фиксируется одной транзакцией: целиком успех или целиком откат; промежуточные состояния извне не видны.
V3 — сравнение и ревью (diff)	Предлагаемое изменение можно представить как разницу к базовой версии и принять или отклонить до попадания в утверждённое состояние (основа утверждения и контрольных точек `QG`).
V4 — ветвление и набор изменений	Незавершённая работа отделена от утверждённого источника истины; несколько независимых изменений ведутся параллельно без влияния на источник истины.
V5 — межсценарная фиксация версии	Можно зафиксировать конкретную версию артефакта из другого носителя как разрешимый идентификатор (основа поля `verifies[].requirement-version`).

V6 — автор и метка времени	Для каждой атомарной правки зарегистрированы однозначный автор и метка времени (основа подписи `ADAPT` и блока `ai-provenance`).

Примеры сред: распределённая VCS (`git` с ревью запросов на слияние), документоориентированное хранилище с цепочкой ревизий и подписями, любая СУБД с историей изменений и подписями. RENAR не требует `git` — только возможности **V1–V6**.

2.8 Происхождение ИИ (`ai-provenance`, канонические поля)

Во `frontmatter` любого артефакта, сгенерированного ИИ:

Поле	Тип	Описание
`ai-provenance.generated-by`	string	Модель в формате `<vendor>-<model>-<version>@<date>`. Пример: `anthropic-claude-opus-4-7@2026-05-15`.
`ai-provenance.prompt-template`	string	Путь к шаблону промпта и версия. Пример: `prompts/adapt-from-tz.md@v2.1`.
`ai-provenance.context-tokens`	integer	Число токенов во входном контексте.
`ai-provenance.output-tokens`	integer	Число токенов в выводе модели.
`ai-provenance.generation-time-ms`	integer	Время генерации в миллисекундах.
`ai-provenance.human-edits`	boolean	Значение «истина», если текст после генерации правил человек. **Для утверждённых артефактов обязательно `true`.**

2.9 Типы контрольных примеров

Закрытый список — шесть типов (каноника по `standard/09 §9.5`):

Тип TC (`tc-type` поле)	ISTQB соответствие	Применение
`acceptance`	Acceptance Testing	Проверка BR (приёмочный).
`ux`	расширение по удобству	Проверка `SPEC-UI` через модель-судью VLM / визуальное сравнение с эталоном (`baseline`).
`system`	System Testing	Проверка SR, SPEC-PROC, SPEC-ARCH.
`contract`	Component Integration Testing	Проверка SPEC-API / SPEC-INT / SPEC-DATA через contract testing (Pact и аналоги).
`eval`	специфично для ИИ	Проверка `SPEC-AI` через оценивающую LLM и метрики (BLEU, точность, доля галлюцинаций).

`security`	расширение по безопасности	Проверка `SPEC-SEC` : инварианты безопасности (STRIDE); нормативно только негативные сценарии (`standard/09 §9.6.4`).

2.10 Связи (frontmatter поля)

Поле	Назначение
`parent`	Родитель в иерархии (BR → SR → TR); один источник.
`children`	Дочерние артефакты (выводятся автоматически).
`source.adapt`	`ADAPT` , из которого выведен артефакт (каноника для `BR` / `SR` / `SPEC`).
`source.adapt-section`	Раздел `ADAPT` (Forward §N).
`source.tz-section`	Раздел ТЗ (справочно, для трассируемости).
`verifies` (в `TC`)	`SR` / `BR` , которые покрывает `TC` , с указанием `requirement-version` .
`verified-by` (в `SR`)	`TC` , подтверждающие `SR` (выводятся автоматически).
`derived-from`	Шаблон и версия (если артефакт создан из шаблона).
`replaces` / `replaced-by`	Замена при статусе `deprecated` .
`supersedes` (в новом)	Какое требование заменяется.
`linked-tasks`	Задачи, реализующие `SR` (через среду выполнения, не через файлы).

2.11 Соглашение об именах файлов (по умолчанию)

Тип	Шаблон	Пример
ADAPT	`adapt/ADAPT-NNN[-delta-N].md`	`adapt/ADAPT-001-main.md` , `adapt/ADAPT-001-delta-1.md`
BR	`br/BR-NN-<slug>.md`	`br/BR-01-notification-capture.md`
SR	`sr/SR-NN-<slug>.md`	`sr/SR-05-notification-feed.md`
SR подсистемы	`sr/<MODULE>-SR-NN.N-<slug>.md`	`sr/WMS-SR-01.2-pick.md`
SPEC-UI	`specs/ui/SPEC-UI-NN-<slug>.md`	`specs/ui/SPEC-UI-02-notification-feed.md`
SPEC-AI	`specs/ai/SPEC-AI-NN-<slug>.md`	`specs/ai/SPEC-AI-01-rag-strategy.md`
SPEC-INT	`specs/int/SPEC-INT-NN-<slug>.md`	`specs/int/SPEC-INT-01-auth-billing.md`
SPEC (generic)	`specs/<type>/SPEC-<KIND>-NN-<slug>.md`	`specs/api/SPEC-API-03-orders.md`
TC	`tests/TC-NN-<slug>.md`	`tests/TC-01-login-success.md`

ТЗ	`tz/TZ-YYYY-NNN.md`	`tz/TZ-2026-001.md`
Дельта-ТЗ	`tz/TZ-YYYY-NNN-delta-N.md`	`tz/TZ-2026-001-delta-1.md`
UX-эталон	`specs/ui/baselines/SPEC-UI-NN-<scenario>.png`	`specs/ui/baselines/SPEC-UI-02-feed-default.png`
Eval dataset	`specs/ai/eval-datasets/SPEC-AI-NN-<slug>.jsonl`	`specs/ai/eval-datasets/SPEC-AI-01-typical-queries.jsonl`

Соглашение по умолчанию; хранилища, встроенные в конкретную среду, могут использовать иную раскладку при условии устойчивых идентификаторов (возможность **V3**).

2.12 Маркеры записи об изменении (справочно)

В носителе, где атомарные изменения сопровождаются метаданными (текст коммита в `git`, описание записи об изменении в документоориентированном хранилище), допустимы маркеры:

Маркер	Назначение
`[delta:TZ-YYYY-NNN]`	Изменение по delta-ТЗ.
`[test-spec-change]`	Изменение критериев типа «пройдено / не пройдено» у `ТС` (отдельное утверждение).
`[baseline-update]`	Обновление эталона UX или набора для оценки (отдельное утверждение).
`[coverage]`	Автоматическая регенерация сводок по покрытию, плану контрольных примеров и требованиям (бот).
`[reconciliation]`	Изменение от агента согласования (reconciliation).
`[multi-model-disagreement]`	Артефакт с расхождением ответов моделей ИИ — требует ручного ревью.
`[AI]`	Префикс для изменений, сгенерированных ИИ.

Механизм, встроенный в носитель, может выразить эти маркеры полями записи об изменении, метками или тегами — детали не нормируются.

3. Сопоставление со стандартами

3.1 Уровни требований

Метки каноники v1.0 RENAR и внешние стандарты:

RENAR	ISO/IEC 29148	BABOK	SAFe	Document store (example enum)	SENAR (RU)
BR	Business Requirement	Business Need	Portfolio Epic / Strategic Theme	`BT`	БТ

SR	System / Software Requirement	Solution Requirement (Functional)	Feature	`ST`	CT
SR (`level: module`)	(область подкомпонента, расширение)	(область подкомпонента)	Story (иногда)	`TM` (legacy)	CT модул
SR (`constrained-by: SPEC-INT-N`)	Требование к интерфейсу	Интерфейсное решение	Межфичевая интеграция	(связано)	INT-CT (legacy
TR	(нет прямого класса; детализация требования к системе / элементу системы)	Детализированное требование к решению	Story	`TK`	T3
TC	Контрольный пример (Test Case)	Верификация	Приёмочный тест истории	`test_case`	TK
TC (`tc-type: contract`)	Тест интерфейса	Component Integration Testing	Контрактный тест	(связано)	INT-TK (legacy
SPEC-UI	(между BR/SR — проектная спецификация)	Требование стейкхолдера (фрагмент UX)	(уровень проектирования)	(расширение)	UIC (legacy
SPEC-AI	(расширение REQ)	(н/д)	Enabler	(расширение)	AIC (legacy
SPEC-ARCH / SPEC-OPS	Описание проектирования	Компонент решения	Enabler tech spec	(расширение)	TC (legacy

Примечание: колонки «document store (пример перечисления)» и «SENAR (RU)» содержат исторические метки (`TM` , `UIC` , `AIC` , `INT-CT` и т.д.) для трассируемости с системами до v1.0. Колонка канониki RENAR — метки v1.0 согласно §2.1.1. При экспорте в document store или в носитель SENAR применяют целевое сопоставление.

3.2 Контрольные точки качества

Сопоставление канонических `QG-N` v1.0 RENAR с внешними моделями. Устаревшие имена `QG` (§14.4.1) в колонке SENAR сохранены для исторической трассируемости.

RENAR (v1.0)	SENAR (сопоставление с legacy)	Document store (пример)	Активность CMMI
QG-0 Контрольная точка утверждения	QG-0 (контекст)	`VK-1` (старт)	Проверка требований до обязательств

QG-1 Контрольная точка реализации	QG-2 (реализация в legacy)	`VK-1`	Базовая линия реализации (готовность `TC`)
QG-2 Контрольная точка проверки	QG-3 (верификация в legacy)	`VK-2`	Верификация
QG-3 Контрольная точка архитектуры *(опционально)*	(н/д)	`VK-3` (частично)	Утверждение архитектурного решения
QG-4 Контрольная точка приёмки *(опционально)*	QG-4 (Приёмка)	`VK-4`	Приёмка заказчиком

Примечание: до v1.0 `QG-1 Requirements Gate` фактически разделён между каноническим `QG-0 Approval Gate` (утверждение `BR` / `SR` / `SPEC`) и `QG-1 Implementation Gate` (готовность `TC`). См. §14.4.1 для полного сопоставления.

3.3 Статусы жизненного цикла

RENAR	Document store (example)	ISO/IEC 29148	CMMI
`draft`	`draft`	proposed	identified
`approved`	`approved`	agreed-to / baselined	committed
`verified`	`verified`	verified	validated
`deprecated`	`obsolete`	retired	obsolete

3.4 Артефакты процесса

RENAR	BABOK	SAFe	SENAR
ADAPT (Forward + Backward)	Requirements Analysis Document	Solution Intent (fixed + variable)	(n/a — RENAR-extension)
Work Order / T3	Stakeholder commitment artifact	Customer order	(контекст)
delta-TZ	Change Request	(n/a — handled via Solution Intent updates)	(контекст)
Impact Analysis	Impact Analysis (BABOK §8)	(derived)	(контекст)
Выборочная проверка	Random sampling QA	(n/a)	Rule 9.5
Состязательный обзор	Independent verification	(n/a — REQ-extension)	(via ADR metric)
Reconciliation	Continuous improvement audit	Inspect & Adapt	Quality Sweep

3.5 Переводы в пользовательском интерфейсе

Поля `frontmatter`, идентификаторы и имена файлов — всегда канонические (латиница). В интерфейсе допустимы русские подписи:

Канонический	UI (RU)
Business Requirement	Бизнес-требование
System Requirement	Системное требование
Test Case	Контрольный пример (`TC`)
Quality Gate	Контрольная точка качества
Acceptance	Приёмка
Verified	Проверено
Approved	Утверждено
Deprecated	Устарело
Frozen	Замороженный
Backward finding	Замечание к ТЗ
Forward interpretation	Инженерная интерпретация

Перевод в интерфейсе не заменяет канонические идентификаторы в носителе.

4. Запрещённые / устаревшие термины

RENAR не использует следующие термины (даже если они встречаются в SENAR / отраслевой литературе):

Термин	Что использовать вместо	Почему
User Story как требование	SR	Story — единица планирования, не требование. Story может реализовывать SR, но сама требованием не является.
Use Case (формально)	SPEC-UI + SR	Use case смешивает UX и поведение. RENAR разделяет SPEC-UI (UX-эталон) и SR (поведение).
Spec (без квалификатора)	SR / BR / SPEC-API / SPEC-DATA / ...	«Spec» неоднозначно. Используем точные термины.
Бизнес-логика как требование	SR	«Бизнес-логика» — термин кода, не требований.
Функциональность	`SR` / `TR`	Слишком широко; не поддаётся однозначной проверке.
Фича (mixed-lang)	Feature (SAFe context) или SR (канонический термин RENAR)	Mixed Russian/English; неоднозначно.

Хотелка	(никогда)	Договорный документ так не пишется.
Эпик как требование	BR (бизнес-уровень) или Portfolio Epic (SAFe)	Epic — единица планирования, не требование.
«Тестировать руками»	выборочная проверка (Rule 5 Core) или ручной TC с `type: acceptance`	Расплывчатое; нет проверяемого свидетельства.
«Доделать» (как статус)	`draft` / `review`	Не из закрытого списка жизненного цикла.
TODO во `frontmatter`	запись backward в `ADAPT` (если речь о требовании) или задача в трекере (если о реализации)	Открытые вопросы живут в нужном артефакте, не в комментарии к полю.

При таких находках в локальных артефактах проекта включайте предупреждение на стороне носителя (`pre-commit` в `git` , правило валидации в document store).

---

5. Версионирование глоссария

Глоссарий — самостоятельный документ со своей версией. Изменение канонического термина — скачок major-версии (1.0 → 2.0) и сценарий миграции для всех проектных артефактов.

Текущая версия: 1.0-reconciled (согласование фазы 1.5 с `standard/04-terms.md §4.14.1`).

5.1 Открытые вопросы — закрыты (фаза 1.5, 2026-05-16)

Четыре открытых вопроса прежней редакции черновика закрыты согласованием с `standard/04-terms.md §4.14.1` :

#	Был открытым вопросом	Итог	Источник
1	Канонический язык: латиница (`BR` / `SR`) или русский (БТ/СТ)?	**Канон — латиница**; русский только в проекции UI (§4.5)	`standard/04 §4.13.3` + `reference/06-ru-style-guide.md §1.3`
2	`TM` как отдельная метка или уточнение `SR` ?	**`SR` с `level: module`** — уточнение, не отдельный тип артефакта	`standard/04 §4.14.1` (устарело `TM`) + §2.1.1
3	AIC ← AAC / AIA / AIC?	**`SPEC-AI`** (каноника v1.0); AIC — устаревшая метка	`standard/04 §4.14.1` + §2.1.1
4	`INT-TC` отдельный тип или соглашение об именах?	**`TC` с `tc-type: contract`** — уточнение, не отдельный тип	`standard/04 §4.14.1` (`INT-TC` deprecated) + §2.1.1

5.2 История согласований

Дата	Версия	Изменение

2026-05-16	1.0-reconciled	Согласование фазы 1.5 с `standard/04-terms.md §4.14.1`. Устаревшие метки перенесены из таблицы `§14.1` в `§14.1.1`; имена `QG` приведены к канонике v1.0 в §14.4; устаревшие имена → `§14.1`. Обновлены таблицы сопоставления §4.1 и §4.2. Четыре открытых вопроса закрыты (§2.1). Задача: `ru-reconcile-glossary-vs-standard`.
(ранние черновики)	1.0-draft	Наполнение черновика в фазе 7; были открытые вопросы и расхождения с `standard/04 §4.14.1`. История коммита зафиксирована; заменено выпуском 1.0-reconciled.

5.3 Перекрёстные ссылки

- **Стиль редакции v1.0** (`reference/06-ru-style-guide.md`) — правила русской редакции нормативного текста; §1.9 задаёт канонический перечень терминов параллельно этому глоссарию. При расхождении приоритет формулировок для редакционных проходов — у руководства по стилю.
- **Канонические определения** (`standard/04-terms.md`); §4.14.1 — сопоставление устаревшие → каноника.

Глоссарий RENAR 1.0-reconciled — часть `reference/`. См. также 02-schemas.md, 03-ai-risk-register.md, 06-ru-style-guide.md.

Схемы (формальные)

> ***Назначение:*** *машино-читаемые YAML frontmatter схемы для всех типов артефактов RENAR. Используются нативными для носителя валидаторами для проверки соответствия.*
> ***Нормативные определения структуры*** *— в* `standard/06`, `standard/07`, `standard/08`, `standard/09`. *Валидация примеров:* `node scripts/validate-schema-examples.js`. *Этот документ — справочник (informative lookup).*

1. Общий frontmatter (все артефакты требований и SPEC)

Поля, общие для BR/SR/TR/SPEC-* (канонические v1.0). Legacy типы `UIC` / `AIC` / `INT-SR` / `TS` deprecated в v1.0 (standard/04 §4.14.1).

```yaml
# Identity
id: "<TYPE>-NN[.N]"
title: "<short, descriptive>"
type: BR | SR | TR | SPEC-ARCH | SPEC-API | SPEC-DATA | SPEC-INT | SPEC-PROC |
SPEC-UI | SPEC-AI | SPEC-SEC | SPEC-OPS
slug: "<kebab-case>"

# Scope
level: system | subsystem | module
scope: { system: "<system-id>", subsystem: "<subsystem-id>" }   # subsystem=null
если level=system

# Жизненный цикл (verified — BR/SR; frozen — ADAPT §7; ready/passing/failing — TC
§8; см. standard/04 §4.6)
status: draft | review | approved | verified | deprecated | obsolete | frozen
priority: must | should | could       # MoSCoW; SAFe — через WSJF (BR-specific)

# Provenance (conditional, standard/07 §7.4.1): ADAPT реактивен.
# Findings present → source.adapt + adapt-section mandatory. No findings →
adversarial-review-ref mandatory.
# source.tz-section — обязательно всегда.
source:
  adapt: "ADAPT-NNN"                  # conditional
  adapt-section: "Forward §N.N"       # mandatory если adapt present
  tz-section: "§N.N"                  # обязательно всегда
  adversarial-review-ref: "<ссылка>"  # mandatory когда adapt omitted
  document-ref: "<ссылка>"            # pinned ревизия source-документа

# Hierarchy
parent: { id: "<parent-id>", ref: "<ссылка>" }   # id required для SR (→BR), TR
(→SR); optional для BR
children: []                          # auto-derived
```

```
# Связь с SPEC (граф)
constrained-by: []                    # SR → SPEC-* (типизированные рёбра)
implements-spec: []                   # TR → SPEC-*
depends-on: []                        # между SPEC

# Verification
verified-by: []                       # auto-derived; TC IDs
verifies-business-goal: ""            # optional

# AI provenance (RENAR-4+ обязательно для approved)
ai-provenance:
  generated-by: "<vendor>-<model>@<date>"
  prompt-template: "<template-path>@<version>"
  context-tokens: integer
  output-tokens: integer
  generation-time-ms: integer
  generated-at: "<ISO-8601>"
  human-edits: boolean                # true required для approved

# AI cost budget (optional)
ai-budget: { context-tokens-target: integer, context-tokens-actual: integer,
output-tokens-target: integer, output-tokens-actual: integer, generation-time-
target-ms: integer }

# Замена + schema versioning
replaces: "<old-id>"
replaced-by: "<new-id>"
deprecated-date: "<ISO date>"
schema-version: "1.0"
```

2. BR — Business Requirement

```
# Extends common §1, дополнительно:

level: system | subsystem             # BR на уровне модуля запрещён (standard/06
§6.4)
scope: { system: "<system-id>", subsystem: "<subsystem-id>" }

# Межуровневая связь BR подсистемы → BR системы (standard/06 §6.8.2)
implements:                             # массив; substrate-agnostic ссылка
  - { id: BR-NN, scope: { system: "<system-id>" }, rationale: "<short>" }   #
rationale опционально
implemented-by: []                      # auto-derived (обратное ребро; не пишется
автором)

business-context:
  stakeholder: "<role>"
  business-goal: "<short statement>"
  kpi-impact:
    - { kpi: "<name>", direction: increase|decrease, target: "<measurable>" }
```

```
# business-outcome — required для QG-4
business-outcome:
  measurement-type: kpi | survey | observation | usage
  kpi-name: "<KPI>"
  measurement-method: "<how>"
  baseline-value: number
  baseline-measured-at: "<ISO date>"
  target-value: number
  target-met-by: "<ISO date>"
  current-value: { value: number, measured-at: "<ISO date>", achievement: "
<percent>" }

prioritization: { framework: WSJF|RICE|MoSCoW, wsjf-score: number, prioritized-
at: "<ISO date>", prioritized-by: "<role>" }

data-classification:
  contains-pii: boolean
  contains-financial: boolean
  contains-health: boolean
  contains-children-data: boolean
  retention-days: integer
  data-residency: ["RU" | "EU" | "US" | ...]

compliance:
  - { standard: "ISO 27001:2022", control: "<id>", rationale: "..." }
  - { standard: "GDPR", article: "Art.NN" }
  - { standard: "ФЗ-152", article: "ст.NN" }

ai-act: { risk-class: prohibited|high|limited|minimal, rationale: "<reason>",
high-risk-domain: boolean }
```

Поля `implements[]` / `implemented-by[]` — нормативные правила

Правило	Уровень
`implements[]` обязателен при `level: subsystem` И когда родительская система имеет ≥1 approved BR	recommended v1.0; обязательно v1.1+
Target BR обязан быть в статусе `approved` или выше на момент approve данного BR	hook-enforced
Cycle detection: цепочка `implements` не должна образовывать циклов	hook-enforced
`implements[]` — **не** parent-edge; запрет множественных parents (standard/06 §6.8.3) распространяется на SR/TR, не на BR	normative
Deprecate target BR → cascade-warning для всех `implemented-by` (не cascade-deprecate)	hook-enforced
Cross-substrate синтаксис: `id + scope.system` не зависит от носителя	normative
Cardinality: array (0..N)	normative

Гейт обеспечения соблюдения — `scripts/check-implements-edge.js`. Поле `implemented-by` — auto-derived; ручная запись запрещена.

3. SR — System Requirement

```
# Extends common §1, дополнительно:

parent:
  id: "BR-NN"                          # required

# Источник ADAPT — через канонические source.adapt / source.adapt-section (§1).
# Отдельного поля derived-from-adapt нет (standard/06 §6.6.2).

constrained-by:                        # типизированные рёбра к SPEC-*
  - "SPEC-UI-NN"
  - "SPEC-API-NN"
  - "SPEC-DATA-NN"
  - "SPEC-SEC-NN"

quality-characteristic:                # ISO/IEC 25010:2023 (9 характеристик;
interaction-capability ← usability, flexibility ← portability в 25010:2011;
safety — новая в 25010:2023)
  - functional-suitability | performance-efficiency | compatibility |
interaction-capability | reliability | security | maintainability | flexibility |
safety

# Inherited from parent BR (если применимо): data-classification, compliance, ai-
act
```

4. TR — Task Requirement

```
# Extends common §1, дополнительно:

parent:
  id: "SR-NN"                          # required

implements-spec: []                    # SPEC-* реализуемые этой задачей
estimated-effort: "<short statement>" # optional, free-form
```

5. SPEC-* common schema

Все 9 типов SPEC делят общую структуру (§1) + следующие SPEC-specific поля:

```
type: SPEC-ARCH | SPEC-API | SPEC-DATA | SPEC-INT | SPEC-PROC | SPEC-UI | SPEC-AI
| SPEC-SEC | SPEC-OPS

referenced-by: []                    # auto-derived
depends-on: []                       # SPEC от которых этот зависит
compliance-refs: []                  # ISO / GDPR / ФЗ-152 / AI Act / NIST AI
RMF
```

Обязательные разделы body: `## Назначение`, `## Scope`, `## <Type-specific sections — см. §6>`, `## Связь с требованиями`, `## Связь с другими SPEC`, `## Verification`, `## Open questions`.

---

6. SPEC type-specific extensions

Type-specific поля для 9 типов SPEC. Industry references — в `standard/14`. Legacy замены: `UIC` → SPEC-UI, `AIC` → SPEC-AI, `INT-SR` → SPEC-INT.

6.1 SPEC-ARCH, SPEC-API, SPEC-DATA, SPEC-INT, SPEC-PROC

```
# SPEC-ARCH
arch-style: monolith | microservices | modular-monolith | serverless | hybrid
deployment-model: cloud | on-prem | hybrid | edge
tech-stack: { languages: [], frameworks: [], data-stores: [], message-brokers: []
}
quality-attributes: [{ name: latency, target: "p95 < 200ms" }, { name:
availability, target: "99.9%" }]

# SPEC-API
api-style: rest | graphql | grpc | websocket | async-events
api-version: "v1.2.0"
versioning-strategy: url-path | header | query-param | content-negotiation
authentication: bearer-jwt | api-key | oauth2 | mtls | none
rate-limits: [{ endpoint: "*", limit: "1000/min/key" }]
contract-file: { format: openapi-3.1 | asyncapi-2.6 | proto3, location:
"contracts/<name>.yaml" }

# SPEC-DATA
data-style: relational | document | graph | columnar | hybrid
storage-engine: postgresql | mysql | couchdb | mongodb | clickhouse | ...
schema-version: "1.4.0"
pii-classification: [{ entity: User, fields: [email, phone], level: PII-high }]
retention-policies: [{ entity: Order, period: "7 years", basis: "tax law" }]
migration-strategy: forward-only | reversible | dual-write

# SPEC-INT
integration-pattern: request-response | event-driven | message-queue | webhook |
file-transfer
```

```
direction: outbound | inbound | bidirectional
counterparty: { system: "<external-name>", contract-owner: "<team-or-vendor>",
contract-ref: "<external-spec-url>" }
sla: { availability: "99.5%", latency-p95: "500ms", fallback: "queue + retry;
manual reconciliation after 24h" }
idempotency: guaranteed | best-effort | none

# SPEC-PROC
process-style: bpmn | state-machine | saga | choreography | orchestration
state-count: integer
participants: [{ role: customer, system: client-portal }, { role: agent, system:
back-office }]
sla: { end-to-end: "2 business hours" }
compensation: defined | not-applicable | manual
```

6.2 SPEC-UI, SPEC-AI, SPEC-SEC, SPEC-OPS

```
# SPEC-UI
ui-platform: web | mobile-ios | mobile-android | desktop | tv | embedded
target-users: [{ role: end-customer, persona: "ADAPT-NNN §X.Y" }]
design-system: "<reference-or-internal>"
accessibility-level: WCAG-A | WCAG-AA | WCAG-AAA
i18n: required | not-required
mockup-links: [{ tool: figma, url: "<link>", version: "v3" }]
baseline-images: ["ai-concepts/baselines/SPEC-UI-NN-screen-01.png"]

# SPEC-AI (judge-model.vendor ≠ production-model.vendor — нормативно)
ai-pattern: rag | fine-tuning | prompt-engineering | tool-use | multi-agent |
embedding-only
production-model: { vendor: anthropic|openai|google|local, model: "<name>",
version: "<exact>" }
judge-model: { vendor: "<different-vendor>", model: "<different-model>" }
context-strategy: { embedding-model: "<model>", chunk-size: integer, chunk-
overlap: integer, vector-store: pinecone|weaviate|pgvector|qdrant }
eval-strategy: { metric: accuracy|f1|rouge|custom-rubric, threshold: number,
baseline-dataset: "<path>" }
cost-budget: { tokens-per-request-target: integer, tokens-per-request-ceiling:
integer, monthly-budget-usd: number }

# SPEC-SEC
security-domains: [authentication, authorization, data-protection, audit,
secrets-management]
auth-model: { authn: jwt-bearer|oauth2-pkce|mtls|passkey, authz: rbac|abac|relbac
}
data-classification: [{ class: PII-high, fields: [...] }, { class: PCI, fields:
[...] }]
threat-model-method: STRIDE | PASTA | OCTAVE
compliance: [ISO-27001, GDPR, Ф3-152, PCI-DSS-4]

# SPEC-OPS
deployment-style: kubernetes | vm | serverless | docker-compose | bare-metal
```

```
environments:
  - { name: dev, purpose: development, scale: minimal }
  - { name: staging, purpose: integration-testing, scale: half-prod }
  - { name: prod, purpose: production, scale: full }
slo: { availability: "99.9%", error-budget-month: "43m", latency-p95: "300ms" }
observability: { logs: elastic|loki|cloudwatch, metrics:
prometheus|datadog|cloudwatch, traces: jaeger|tempo|x-ray }
disaster-recovery: { rto: "<duration>", rpo: "<duration>" }
```

7. ADAPT schema

ADAPT — отдельный артефакт (`standard/07`). Реактивный: существует только при findings от состязательного обзора ТЗ (§7.4.1). Вердикт «no findings» — ADAPT не создаётся, фиксируется через `<artifact>.source.adversarial-review-ref` .

```
# Identity
id: ADAPT-NNN
title: "Адаптация ТЗ <name>"
type: ADAPT
trigger-stage: import-tz | decompose-br | decompose-sr | spec | tc   # стадия-
триггер (standard/07 §7.4.1.4)

# Source
source-tz: { id: TZ-YYYY-NNN, signed-date: "<ISO-date>", signed-by-client: "
<name-role>", document-ref: "<ссылка>" }
parent-adapt: { id: ADAPT-NNN, delta-tz: TZ-YYYY-NNN }   # для delta-ADAPT

# Supersession (standard/07 §7.6.4) — только для superseding-ADAPT
supersedes: ADAPT-MMM                                    # ссылка на
дезавуируемый ADAPT
superseded-by: ADAPT-NNN                                 # auto-derived; на
дезавуируемом
supersession-rationale: "<противоречащее BR/SR/SPEC + источник>"   # mandatory
если supersedes присутствует

# Lifecycle (подмножество §1: ADAPT не использует verified/deprecated; superseded
— терминальное при дезавуировании)
status: draft | review | client-ready | answered | approved | frozen | superseded
| obsolete
created: "<ISO-date>"
last-updated: "<ISO-date>"

# Approval (required для approved)
approval:
  client-signature: { signed-by: "<name>", role: "<role>", organization: "
<client-org>", signed-at: "<ISO-datetime>", signature-ref: "<ссылка>" }
  architect-signature: { signed-by: "<name>", role: architect, signed-at: "<ISO-
datetime>" }

# Auto-derived
```

```
generates-requirements: []
generates-specs: []
open-questions-count: integer          # должен быть 0 для approved
resolved-questions-count: integer

# AI provenance
ai-provenance: { generated-by: "<vendor>-<model>@<date>", prompt-template: "
<template-path>@<version>", context-tokens: integer, output-tokens: integer,
human-edits: boolean }
```

Backward записи внутри body:

```
id: B-NNN
category: contradiction | gap | hidden-assumption | feasibility | regulatory |
terminology | scope
status: open | asked-to-client | answered | resolved | frozen
tz-section: "§N.N"
description: "..."
asked-to-client: "<ISO-date>"
client-answer:
  signed-by: "<name>"
  signed-at: "<ISO-datetime>"
  channel: email | docusign | zoom-transcript | written-letter
  text: "..."
resolution: "..."                      # как ответ интегрирован в Forward
```

8. TC — Test Case

```
# Identity
id: "TC-NN[.N]"
title: "<descriptive>"
type: TC
slug: "<kebab-case>"

# Classification
tc-type: acceptance | ux | system | contract | eval | security
negative: boolean                      # true для парного негативного TC

# Scope
level: system | subsystem | module
scope: { system: "<system-id>", subsystem: "<subsystem-id>", module: "<module-
id>" }

# Lifecycle
status: draft | ready | passing | failing | obsolete

# Verification mapping (≥1)
verifies:
```

```
  - { id: "<requirement-id>", ref: "<ссылка>", requirement-version: "<version-ref>" }   # V5 pinning (standard/03 §3.3.5)

# Pair link (mandatory если negative=false и существует парный)
paired-with: ["<TC-id>"]

# Automation
automation:
  status: automated | manual-pending
  location: "<path-to-implementation>"      # mandatory если automated
  runner: pytest | jest | go-test | playwright | vlm-judge | ragas | pact | other
  manual-pending-until: "<ISO date>"        # mandatory если manual-pending
  manual-pending-reason: "<why>"

# Execution (mandatory если tc-type=ux | eval; judge.vendor ≠ production model vendor — см. §6.2 SPEC-AI, §9)
judge: { vendor: "<provider>", model: "<model-id>", prompt-template: "<template-path>@<version>" }
baseline: { artifact: "<pointer>", perceptual-diff-threshold: float, metric-thresholds: {} }

# Last run (auto-managed; bot-only)
last-run:
  date: "<ISO timestamp>"
  result: pass | fail | skipped | n/a
  runner-id: "<runner@version>"
  run-ref: "<ссылка>"
  requirement-version: "<version-ref>"
  judge-report: "<for ux/eval>"

# Замена / obsolescence
obsolete-pending: boolean             # true при detected delta-T3 инвалидации
replaces: "<old-id>"
replaced-by: "<new-id>"
obsoleted-date: "<ISO date>"

# Inherited
ai-provenance: { ... }                # см. §1
```

9. Validation rules (cross-field)

Правила, не выражаемые в чистой JSON Schema; требуют custom validator. Колонка «Формальная проверка» даёт исполнимый предикат или ссылку на готовый KG-запрос (reference/05 §5/§6).

Правило	Описание	Формальная проверка
ID неизменяем	При изменении файла поле `id` не меняется.	`diff(prev.id, curr.id) == ∅`
`parent` exists	Для SR — parent BR существует и в статусе ≥ `approved`.	`status(BR[SR.parent.id]) ≥ approved`; orphans — 05 §6.1

`source.adapt` approved	Для BR/SR/SPEC — ADAPT в `source.adapt` в статусе `approved` / `frozen`.	`status(ADAPT[art.source.adapt]) ∈ {approved, frozen}`
`verified-by` consistency	TC в `verified-by` имеют `verifies[].id` = этот артефакт.	`∀ tc ∈ art.verified-by: art.id ∈ tc.verifies[].id`
`requirement-version` lock	`TC.last-run.requirement-version` = `verifies[].requirement-version`.	`tc.last-run.requirement-version == tc.verifies[].requirement-version`; stale — 05 §4.4
`source.adapt` для BR/SR/SPEC (conditional)	Канонический источник ADAPT при наличии findings; при вердикте «no findings» — `source.adversarial-review-ref` (standard/07 §7.4.1). TR — через parent SR (standard/06 §6.6.2).	`art.type ∈ {BR, SR, SPEC-*} ⇒ art.source.adapt ≠ null ∨ art.source.adversarial-review-ref ≠ null`
SPEC-AI requires ai-act	Для AI-артефакта `ai-act.risk-class` обязательно.	`art.type == SPEC-AI ⇒ art.ai-act.risk-class ≠ null`
Data residency consistency	RU в `SR.data-classification.data-residency` ⇒ то же в parent BR.	`'RU' ∈ SR.…data-residency ⇒ 'RU' ∈ BR[SR.parent].…data-residency`
Compliance hierarchy	`SR.compliance ⊆ parent BR.compliance` (или явный justification).	`SR.compliance ⊆ BR[parent].compliance ∨ exists(extension-justification)`
TC `automated` requires location	`automation.status: automated` ⇒ `automation.location` непустое.	`tc.automation.status == 'automated' ⇒ tc.automation.location ≠ ''`
Negative TC обязателен	На каждое нормативное утверждение — TC с `negative: true`.	`∀ assertion ∈ art: ∃ tc(negative: true)` (standard/09 §9.7)
ADAPT open-questions == 0 for approved	Approval блокируется при `open` / `asked-to-client` backward.	`adapt.status == approved ⇒ count(backward[status ∈ {open, asked-to-client}]) == 0` (05 §4.7)
Дезавуирование корректно	`supersedes` ⇒ непустой `supersession-rationale` и симметричный `superseded-by` на цели; нет висячих `source.adapt` на `superseded` (standard/07 §7.6.4, standard/10 §10.8.5).	`adapt.supersedes ≠ null ⇒ adapt.supersession-rationale ≠ '' ∧ ADAPT[adapt.supersedes].superseded-by == adapt.id`; `∄ art: art.source.adapt = X ∧ status(ADAPT[X]) == superseded`
Judge isolation (SPEC-AI)	`judge.vendor ≠ production-model.vendor`.	`tc.judge.vendor ≠ SPEC-AI[tc.verifies].production-model.vendor`

SPEC depends-on acyclic	Граф `depends-on` между SPEC — DAG.	cypher cycle-detection (05 §4.6): rows ≠ ∅ ⇒ нарушение

10. Изоморфизм носителя

Отображение для git (YAML frontmatter) ↔ document-oriented store (JSON document):

Поле (canonical)	git (YAML frontmatter)	Document store (JSON doc)
`id`	`id`	`_id = <project>:<doc-type>:<slug>`, поле `slug`
`type: BR`	`type: BR`	`level: "business"`
`type: SPEC-API`	`type: SPEC-API`	`doc_type: "spec_api"`
`parent.id`	`parent.id`	`parent`
`children`	(auto-derived)	`children` (auto-derived)
`status`	`status`	`status`
`priority`	`priority`	`priority`
`source.adapt`	`source.adapt`	`created_from_adapt`
`constrained-by[]`	`constrained-by`	`constrained_by` (subdoc array)
`verified-by[]`	(auto-derived)	`linked_tests`
`ai-provenance.*`	nested object	nested subdocument
`compliance`	`compliance` array	`compliance` subdoc
`data-classification`	nested object	nested subdoc
`business-outcome`	nested object	nested subdoc
`replaces` / `replaced-by`	string ID	`replaces` / `replaced_by`

Нативные для носителя имена полей могут отличаться, но **семантика и invariants сохраняются** через capabilities V1-V6 (01-glossary.md §27).

11. Schema versioning

Каждый артефакт имеет поле `schema-version` (semver). При несовпадении версии файла и текущей схемы validator предлагает migration.

Изменение	Bump
Новое необязательное поле	minor (1.0 → 1.1)

Новое обязательное поле	major (1.0 → 2.0) + migration script
Удаление поля	major + migration script
Изменение enum	minor если добавление, major если удаление
Переименование поля	major + migration script

Текущая версия schemas: 1.0-draft.

12. JSON Schema fragment example (BR)

Ключевые patterns (полная BR-схема — `reference/schemas/br.json`, планируется):

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "https://renar.tech/schemas/br.json",
  "type": "object",
  "required": ["id", "title", "type", "status", "priority", "source", "ai-
provenance"],
  "properties": {
    "id":     { "type": "string", "pattern": "^BR-[0-9]{2}(\\.[0-9]+)?$" },
    "title":  { "type": "string", "minLength": 5, "maxLength": 100 },
    "type":   { "const": "BR" },
    "status": { "enum": ["draft", "review", "approved", "verified", "deprecated",
"obsolete"] },
    "priority": { "enum": ["must", "should", "could"] },
    "source": { "type": "object", "required": ["tz-section"], "properties": {
"adapt": { "pattern": "^ADAPT-[0-9]{3}(-delta-[0-9]+)?$" } } },
    "ai-provenance": { "required": ["generated-by", "generated-at"],
"properties": { "generated-by": { "pattern": "^[a-z]+-[a-z0-9-]+@[0-9]{4}-[0-9]
{2}-[0-9]{2}$" } } }
  }
}
```

Аналогичные JSON-схемы для SR/TR/SPEC-*/TC/ADAPT — в `reference/schemas/` (планируется).

Schemas reference RENAR 1.0-draft — см. также 01-glossary.md, `standard/06` - `09` для нормативных определений.

AI Risk Register для RENAR-проектов

> ***Назначение:*** *реестр AI-специфичных рисков для проектов, использующих RENAR (где AI генерирует требования, спецификации и тесты). Основан на ISO/IEC 23894:2023 (AI Risk Management, Annex A risk sources + Clause 6 process по ISO 31000) и NIST AI RMF 1.0. Нормативные mitigation hooks — standard/09 §9.4, standard/07 §7.10.*

Не подменяет общий security risk register организации. AI-риски — отдельный класс из-за специфики генерации и непредсказуемости моделей.

1. Структура реестра

Каждый риск имеет:

```
id: AIR-NN                              # immutable
name: "<short name>"
category: hallucination | injection | drift | bias | sgnl-failure | data-quality
| adversarial | privacy
# enum-значение — часть до скобки; уточнитель в скобках опционален (напр. "sgnl-
failure (process)")
severity: critical | high | medium | low
likelihood: high | medium | low
iso-23894-ref: "§N.N"
nist-rmf-function: govern | map | measure | manage
mitigations:
  - { mechanism: "<description>", enforced-by: "<who/what>", automated: true |
false }
status: active | mitigated | accepted | monitoring | out-of-scope
owner: "@role"
last-reviewed: "YYYY-MM-DD"
related: ["<core rule N>", "<standard chapter>", "<other AIR-NN>"]
```

Список AIR-01..AIR-14 закрыт; новые риски — только через изменение полного RENAR Standard.

Category ↔ NIST AI RMF trustworthiness characteristics (для проверяемости заявления «основан на NIST AI RMF»):

category	NIST AI RMF trustworthiness characteristic
`hallucination` / `data-quality`	Valid & Reliable
`injection` / `adversarial`	Secure & Resilient
`drift`	Valid & Reliable; Safe
`bias`	Fair — with Harmful Bias Managed
`sgnl-failure`	Safe; Accountable & Transparent

`privacy`	Privacy-Enhanced

> ***Ссылки ISO/IEC 23894:2023.*** *Категории AI-рисков в 23894:2023 расположены в* ***Annex A*** *(risk sources); Clause 6 описывает процесс риск-менеджмента (по ISO 31000). Дескрипторы «Annex A — …» в реестре — risk-source labels; точное сопоставление с пунктами Annex A подлежит сверке при формальном claim.*

2. Реестр AIR-01..AIR-14

Метаданные всех 14 рисков (Sev=Severity, Like=Likelihood):

AIR	Name	Category	Sev	Like	ISO 23894 (Annex A)	NIST RMF	Status
01	Hallucination в AI-генерируемых требованиях	hallucination	**High**	High	Output reliability	Measure	active → mitigate зрелых RENAR levels
02	Prompt injection через T3 от клиента	injection	**High**	Low-Medium	Adversarial inputs	Manage	monitori
03	Model drift / version change	drift	Medium	High	Model жизненный цикл	Manage	monitori
04	Bias в AI-генерации требований	bias	Medium	Medium	Fairness	Measure	active
05	Single-model failure (no diversity)	sgnl-failure	Medium	High	Single point of failure	Manage	mitigate при пол pipeline
06	Test fitting / зеленение тестов	sgnl-failure	**High**	Medium	Verification integrity	Measure	mitigate при выборо провер
07	Hallucinated citations	hallucination	Medium-High	Medium	Output reliability	Measure	monitori
08	Adversarial inputs в clients data (runtime)	adversarial	**High**	Low	Adversarial inputs	Manage	out-of-scope (applicat level)
09	Privacy leakage через AI logs	privacy	**High**	Medium	Privacy	Govern	active
10	Knowledge graph poisoning	data-quality	Medium	Low	Data integrity	Map	monitori

11	Reconciliation false-positive overload	sgnl-failure (process)	Low-Medium	Medium	Verification integrity	Manage	monitori
12	Cost runaway (uncontrolled AI spend)	sgnl-failure (operational)	Medium	Medium	Cost governance	Manage	active
13	Стейкхолдер не понимает AI-сгенерированные требования	data-quality (UX)	Medium	Medium	Transparency	Govern	active
14	Vendor lock-in to specific LLM provider	sgnl-failure (operational)	Medium	Low-Medium	Vendor risk	Govern	monitori

Описание + воздействие + mitigations — ниже.

AIR-01. Hallucination в AI-генерируемых требованиях

AI-агент при генерации BR/SR/SPEC может «дописать от себя» утверждения, которых нет в ADAPT или ТЗ. Воздействие: scope creep, dispute на acceptance, фичи которые не требовались клиентом. **Меры смягчения:** source citation (RENAR Core Правило 1 — каждое утверждение ссылается на ADAPT §N); состязательный обзор (критик-модель другого vendor); метрика Hallucination Rate ≤1% на зрелых уровнях RENAR.

AIR-02. Prompt injection через ТЗ от клиента

Злонамеренный клиент может вставить в ТЗ скрытые инструкции для AI («ignore previous instructions and …»). Воздействие: утечка данных, вредоносные изменения в требованиях, нарушение security policy. **Меры смягчения:** sandboxing AI-агента при импорте (модель работает с ТЗ как с пассивными данными, явно в system prompt); input gateway проверяет на known injection patterns; suspicious pattern → escalation, не auto-process.

AIR-03. Model drift / version change

Anthropic / OpenAI / Google обновляют модели — тот же prompt с тем же ТЗ через 6 месяцев может дать другой output. Воздействие: inconsistency между требованиями в одном проекте; невозможность точно воспроизвести генерацию старого артефакта. **Меры смягчения:** model versioning в `ai-provenance.generated-by` (точная версия + дата); eval-тесты для SPEC-AI прогоняются при смене модели; при регенерации — diff против старой версии и оценка изменений.

AIR-04. Bias в AI-генерации требований

Модель имеет training-data bias — при генерации BR может игнорировать stakeholders определённых групп (accessibility users, non-English locales, специфичные регуляторики). Воздействие: требования не покрывают весь spectrum пользователей; продукт дискриминационный или non-compliant. **Меры смягчения:** multi-model agreement для `priority=must` BR (разные модели — разные biases); карта заинтересованных сторон обязательна в BR (explicit перечисление); состязательный критик с prompt «check for missing stakeholders / accessibility considerations».

AIR-05. Single-model failure (no diversity)

Если все артефакты генерируются одной моделью, её ошибки систематически проникают. «Галлюцинирует» определённый паттерн — все требования это унаследуют. Воздействие: систематическое искажение требований по проекту. **Меры смягчения:** multi-model для `priority=must` BR; состязательный критик с другой моделью; изоляция judge-модели от production-модели в eval-тестах (SPEC-AI: `judge-model.vendor ≠ production-model.vendor`, см. 02-schemas.md §6.2).

AIR-06. Test fitting / зеленение тестов

AI-агент имеет тривиальный путь зеленения failing-теста — ослабить Pass-критерий. Это проходит code review, поскольку «тест зелёный». Воздействие: false confidence; defects проходят в production. **Меры смягчения:** маркер `[test-spec-change]` обязателен для изменения Pass/Fail (отдельный approval); выборочная проверка 5 случайных passing TC раз в спринт (RENAR Core Правило 5); метрика Test-fitting drift rate (отдельная от обычных metrics).

AIR-07. Hallucinated citations

AI-агент пишет citation `[TZ-2026-001 §4 line 142]`, но в реальном ТЗ §4 line 142 — про другое. Citation выглядит как свидетельство, но свидетельство ложное. Воздействие: source citation становится фикцией; цепочка прослеживаемости рвётся при аудите. **Меры смягчения:** citation validator hook (парсит citation, открывает указанный документ, проверяет соответствие); pre-commit/pre-approval блокировка при невалидной citation.

AIR-08. Adversarial inputs в clients data (runtime)

Клиент отправляет данные (через формы, API), специально сконструированные для манипуляции AI-компонент в runtime (не на этапе генерации требований). Воздействие: аналогично AIR-02, но в production runtime. **Меры смягчения:** input sanitization на API gateway; constrained generation (structured outputs only); rate limiting per user. **Status: out-of-scope** — application-level security; RENAR требует SR-уровень покрытия (SPEC-SEC threat model), но runtime защита — задача реализации, не нормирования требований.

AIR-09. Privacy leakage через AI logs

AI-агент при генерации артефакта имеет в контексте PII (из ТЗ или интервью). Логи генерации (tool events, audit records) могут хранить эти PII. Воздействие: PII попадают в логи, в `ai-provenance`, в training data (если используется). **Меры смягчения:** PII redaction в промптах перед отправкой в LLM; `data-classification` tracking; disable training on conversations (Anthropic/OpenAI privacy settings, DPA); TTL на event logs с PII.

AIR-10. Knowledge graph poisoning

Если KG используется как primary search для AI-агентов, incorrect edge может «отравить» все последующие AI-запросы, опирающиеся на этот граф. Воздействие: AI генерирует требования на основе wrong context, систематически. **Меры смягчения:** KG derived от frontmatter, не редактируется

напрямую (см. 05-knowledge-graph-schema.md); CI-валидация графа на каждое изменение (нет circular dependencies, no orphan approved); reconciliation-агент проверяет integrity еженедельно.

AIR-11. Reconciliation false-positive overload

Reconciliation-агент при слишком чувствительных правилах генерирует много false-positive находок. Архитектор начинает их игнорировать → реальные находки тонут. Воздействие: reconciliation теряет ценность, дисциплина не масштабируется. **Меры смягчения:** tunable thresholds в проектной конфигурации; метрика Reconciliation Findings/Week (если растёт без real issues — re-calibration); архитектор может отклонять находки с обоснованием — feedback для tuning prompt агента.

AIR-12. Cost runaway (uncontrolled AI spend)

Без budget tracking AI-генерация (особенно с multi-model, состязательный, eval) может потреблять токены непропорционально размеру проекта. Воздействие: финансовые потери; нерентабельность практики. **Меры смягчения:** `ai-budget` field в frontmatter (target + actual); aggregated cost metric на уровне проекта; cap на проект; alarm при approached; recommendation engine «Sonnet/Haiku для рутины, Opus только для `priority=must` BR».

AIR-13. Стейкхолдер не понимает AI-сгенерированные требования

AI генерирует SR в техническом стиле; клиент / нетехническая заинтересованная сторона при рецензировании не понимает → утверждение становится формальностью. Воздействие: QG-ADAPT-approve / QG-4 acceptance теряет смысл; dispute rate at acceptance растёт. **Меры смягчения:** style guide (04-ai-style-guide.md); BR в business language (технологии — в SPEC-*, не в BR); human-readable summary в каждом BR/SR — короткая секция, понятная без технического background.

AIR-14. Vendor lock-in to specific LLM provider

Все промпты оптимизированы под конкретного провайдера (Anthropic Claude). Если provider меняет pricing/availability — переход требует переписывания всех промптов. Воздействие: operational risk, costs, business continuity. **Меры смягчения:** provider-agnostic prompts где возможно (избегать vendor-specific tool syntax); multi-model уже enforced для `priority=must` (Принцип 4) — гарантирует второй провайдер в pipeline; periodic test runs на резервном провайдере.

---

3. Risk matrix

```
Severity / Likelihood
          │  Low      Medium          High
──────────┼──────────────────────────────────
High      │  AIR-08   AIR-04, 06      AIR-01, 03
Medium    │  AIR-10   AIR-11, 13      AIR-07, 09, 14
Low       │  —        AIR-12          AIR-02, 05
```

Critical и High риски в top-right квадранте — приоритет mitigation.

---

4. Mitigation matrix

Какие mitigations покрывают какие риски (компенсирующие механизмы: одиночный mitigation редко достаточен; высокие риски требуют ≥ 2 независимых механизмов):

Mitigation	Покрывает риски
Source citation (Core Правило 1)	AIR-01, AIR-07
Состязательный обзор (другая модель)	AIR-01, AIR-04, AIR-05
Выборочная проверка passing TC (Core Правило 5)	AIR-06
Multi-model для `priority=must`	AIR-04, AIR-05, AIR-14
Judge isolation в SPEC-AI	AIR-05
AI-происхождение (model + version + date)	AIR-03, AIR-09
Citation validator hook	AIR-07
Input sandbox / sanitization	AIR-02, AIR-08
PII redaction + DPA	AIR-09
KG validation in CI	AIR-10
Reconciliation tunable thresholds	AIR-11
`ai-budget` field + project cap	AIR-12
Style guide + business-language BR	AIR-13

5. Operational governance

Периодичность рецензирования. Monthly: AIR-01, AIR-02, AIR-06, AIR-07, AIR-09 (high-impact runtime risks). Quarterly: остальные. On-incident: любой риск, в который произошёл инцидент → root cause → mitigation.

Owner. Default — Архитектор проекта. Для AI-специфичных рисков — AI Governance Lead (если есть в организации).

Storage. Risk register проекта — отдельный артефакт:

```
<project>.req/
  governance/
    ai-risk-register.md          # snapshot этого reference + project-specific
notes
    review-log.md                # история ревью с датами и подписями
```

На носителе, не поддерживающем директории — эквивалент namespacing.

Reconciliation-агент. Еженедельный run обновляет `status` и `last-reviewed` поля каждого AIR. Если статус меняется (e.g., monitoring → active) — alert архитектору.

6. Связь со стандартом

AIR	RENAR Core / Standard
AIR-01, 07	Core Правило 1 (ADAPT перед SR) + Standard ch.5
AIR-04, 13	Standard ch.4 (роли) + style guide §04
AIR-05	Standard ch.13 (AI generation) + SPEC-AI judge isolation
AIR-06	Core Правило 4 + Правило 5 + QG-2 Verification Gate
AIR-09	Standard ch.11 compliance + SPEC-SEC
AIR-12	Standard ch.13 (cost governance)

7. Что НЕ покрывает risk register

Этот реестр focuses на AI-специфичные риски процесса RENAR. **Не подменяет:** общий security risk register организации (ISO 27001); compliance risk register (06-compliance.md); application-level threat model конкретного проекта (SPEC-SEC). Reg-обязательные требования регуляторов (AI Act high-risk, ФЗ-152) — отдельные artifacts; этот register — operational tier.

AI Risk Register RENAR 1.0-draft — renar.tech

Руководство по AI-стилю — генерация RENAR-артефактов

> ***Назначение:*** *style, тон, структура, длина и лексикон для AI-агентов, генерирующих RENAR-артефакты (ADAPT, BR, SR, SPEC, TC, Impact Analysis). Снимает класс рисков «разные модели → разный стиль → drift восприятия» (см. AIR-04, AIR-13 в 03-ai-risk-register.md). Дополняет reference/06 RU Style Guide для human editors.*

Контекст: AI-агент в RENAR — **штатный primary author** артефактов (standard/00 §0.2.1); распределение ролей AI vs Архитектор — standard/05 (§5.2.1, §5.3.2, §5.6 RACI). Данный документ нормирует стиль для штатного режима, не для исключений.

Аналог в индустрии: Google Developer Documentation Style Guide, Microsoft Style Guide for technical writing, Diátaxis. Цель — единый ровный тон независимо от модели или промпт-инженера.

---

1. Принципы

1.1 Стиль = договорная точность

RENAR-артефакт — **договорной документ** между клиентом, командой и системой. Никаких метафор, никакого «красочного» языка, никаких ассоциаций. Только однозначные утверждения.

Хорошо:

> *Система должна принимать заявку на регистрацию через POST /auth/register.*

Плохо:

> *Пользователи смогут легко и удобно создавать аккаунты через современный API.*

1.2 Один артефакт = одна мысль

Каждый BR / SR / SPEC / TC покрывает **одну** концепцию. Если AI-агент склонен «упаковать» 3 темы в один SR — он должен декомпозировать, не упаковывать.

Сигнал упаковки: союз «и» в title, multiple actions в одном утверждении.

Плохо:

> *SR-05: Регистрация, авторизация и восстановление пароля*

Хорошо:

> *SR-05: Регистрация клиента SR-06: Авторизация клиента SR-07: Восстановление пароля*

1.3 Нет претензий на полноту

AI не должен «дорабатывать» требование от себя, если в источнике (ADAPT или T3) этого нет. Каждое утверждение либо имеет citation, либо помечено `derived` с явным обоснованием.

Плохо:

```
- При регистрации проверяется уникальность email.
- При регистрации проверяется уникальность телефона.   ← в ADAPT нет, AI домыслил
```

Хорошо:

```
- При регистрации проверяется уникальность email. [ADAPT-001 §14.1 Forward]
```

Если AI считает, что нужна проверка телефона — он не пишет её в SR, а добавляет в Critic-output: «возможно нужна проверка уникальности телефона — нет в ADAPT, требует backward finding к Заинтересованной стороне».

1.4 Однозначность важнее красоты

Если выбор между точным длинным предложением и компактным неоднозначным — AI выбирает точное длинное. Длина — не enemy, ambiguity — enemy.

---

2. Структура и длина по типу артефакта

2.1 ADAPT

Body: 200-800 строк (зависит от объёма ТЗ).

Обязательные секции:

- `## Краткое содержание` — 3-5 параграфов для одностраничного чтения клиентом.
- `## Term mapping` — таблица «клиент → инженер».
- `## Forward: интерпретация по разделам ТЗ` — раздел на каждый § ТЗ.
- `## Backward: обнаруженные проблемы` — записи B-NNN с жизненным циклом.
- `## Резюме backward findings` — таблица по категориям.
- `## Generated artifacts` (auto после approval).

Тон: двусторонний — Forward в инженерных терминах с явной интерпретацией, Backward в клиент-friendly формулировках.

2.2 BR — Business Requirement

Body: 30-80 строк.

Обязательные секции:

- `## Потребность` — одно предложение по шаблону `[Роль] должен [действие], чтобы [бизнес-цель].`
- `## Критерии успеха` — 3-7 измеримых пунктов.
- `## Контекст` — 5-15 строк.

Запрещено:

- Технологии («через REST API», «Postgres», «React»).
- Конкретные экраны или поля БД.
- Размышления автора («мы думаем», «возможно», «было бы хорошо»).

Тон: формальный, императивный («должен», не «может»).

2.3 SR — System Requirement

Body: 40-150 строк.

Обязательные секции:

- `## Описание` — одно предложение по шаблону `Система должна [поведение]. [Условие].`
- `## Поведение` — 5-20 пунктов с inline citations к ADAPT.
- `## Ограничения` — 3-10 пунктов.

Опциональные секции (когда применимо):

- `## Не входит в это требование` — explicit out-of-scope.
- `## Связанные SR` — cross-links.

Запрещено:

- Названия таблиц БД, конкретных функций, классов.
- Frameworks («через FastAPI», «React-компонент»).
- Конкретные UI-paths («кнопка в правом углу») — это в SPEC-UI.

Тон: формальный, точный, поведенческий («система отвечает 422», не «возвращается ошибка»).

2.4 SPEC-* (9 типов)

Body: 80-400 строк (varies по типу).

Общие обязательные секции (см. 02-schemas.md §2):

- `## Назначение`
- `## Scope` (входит / не входит)
- `## <Type-specific sections>`
- `## Связь с требованиями`
- `## Связь с другими SPEC`
- `## Verification`
- `## Open questions`

Тон по типам:

- **SPEC-ARCH / API / DATA / INT / SEC / OPS:** технико-аналитический. Допустимы конкретные технологии.
- **SPEC-UI:** пользователь-центричный нарратив с поведенческой точностью. «Менеджер видит ленту заказов, нажимает на заказ, открывается детальный экран», не «UI должен иметь listing component».
- **SPEC-AI:** model card style — capabilities, limits, failure modes, eval criteria.

- **SPEC-PROC:** procedural narrative с BPMN / машина состояний references.

2.5 TC — Test Case

Body: 30-80 строк.

Обязательные секции:

- `## Контекст` — на какой пункт верифицируемого артефакта ссылается TC.
- `## Предусловия` (Given) — 2-5 строк.
- `## Шаги` (When / действие) — 1-3 строки.
- `## Pass-критерий` (Then / Pass) — бинарный, наблюдаемый, воспроизводимый критерий.
- `## Fail-критерий` — список наблюдаемых признаков нарушения, **включая** возможные побочные эффекты (утечки безопасности, отсутствующие журналы аудита).
- `## Постусловия` — ожидаемое состояние после прогона.
- `## Out of scope` — что намеренно не проверяется, с pointer на покрывающие TC.

> *Имена секций критериев (`## Pass-критерий` / `## Fail-критерий`) — канонические machine-detectable заголовки (standard/09 §9.4), детектируются хуком обеспечения соблюдения (standard/10 §10.11.3); не переименовывать.*

Тон: исполнительный, чёткий («выполняется X», «возвращается Y»).

2.6 Impact Analysis

Body: 50-200 строк (зависит от размера дельты).

Обязательные секции:

- `## Затронутые требования` — таблица с типом изменения.
- `## Затронутые SPEC` — таблица с действием.
- `## Затронутые TC` — таблица с действием.
- `## Затронутые задачи в backlog`.
- `## Open questions` — что нужно уточнить с Заинтересованной стороной через backward в delta-ADAPT.

Тон: аналитический, без оценок. Не «это плохое изменение», а «это изменение затрагивает X требований и Y задач».

---

3. Лексикон

3.1 Use (canonical)

- «Система должна …» (модальность).
- «должен» / «не должен» (RFC 2119 vocabulary — MUST / MUST NOT).
- «должна» / «не должна» (для системы).
- «возвращает» (HTTP/API behavior).
- «предусловия» / «постусловия» (формальные термины тестирования).

- «требование» / «утверждение» / «assertion».
- «критерий приёмки» (acceptance criterion).
- «backward finding» / «forward интерпретация» (RENAR canonical из ADAPT).

3.2 Запрещено

Слово/фраза	Заменить на	Почему
«Хочется чтобы»	«Система должна»	Модальность должна быть строгой.
«Удобно»	конкретный измеримый критерий	Subjective.
«Современный»	конкретные tech requirements в SR/SPEC	Marketing language.
«Качественный»	измеримые quality characteristics из ISO/IEC 25010	Subjective.
«Быстрый»	p95/p99 latency с числом	Vague.
«Надёжный»	uptime / reliability с числом	Vague.
«Возможно» / «Может быть»	(никогда в требованиях)	Hedging.
«Скорее всего»	(никогда)	Same.
«Желательно»	`priority: should` / `could` во frontmatter	Mixing channels.
«Понятный»	criteria for clarity (Flesch reading ease, etc.)	Subjective.
«Гладкий experience»	UX criteria в SPEC-UI	Marketing.
«Бесшовный»	конкретный технический критерий	Marketing.
«Поддерживать» (без объяснения как)	конкретный SR-уровень criteria	Vague.
«Интегрироваться с X» (без contract)	SPEC-INT с counterparty	Vague.

3.3 Use Cases vs Stories vs Requirements

RENAR использует **только каноническую терминологию** из 01-glossary.md. AI-агент **не имеет права** называть требование «User Story», «Use Case», «Scenario», «Capability» — это запрещённые синонимы. Полный список запрещённых терминов и подстановок — в 01-glossary.md §5.

4. Шаблоны утверждений

4.1 BR-утверждение

Шаблон: `[Роль]` должен `[действие]`, чтобы `[бизнес-цель]`.

Пример:

> *Пользователь должен автоматически получать и хранить уведомления выбранных приложений в фоновом режиме, чтобы не пропустить важные сообщения.*

4.2 SR-утверждение

Шаблон: `<actor>` `<action>` `<condition>`. `<consequence>`.

Пример:

> *Система при первом запуске проверяет наличие разрешения NotificationListenerService. Если разрешение не выдано — отображается экран онбординга с кнопкой «Выдать доступ».*

4.3 TC Pass-критерий

Шаблон: `<actor>` `<action>` `<input>`. `<observable>` `<measurement>`.

Пример:

> *POST /auth/login с body `{"email":"x@y.z", "password":"correct"}`. Response status = 200, body содержит JWT с `exp = now + 24h ± 1m`.*

4.4 TC Fail-критерий

Список наблюдаемых признаков нарушения, **включая** возможные побочные эффекты:

```
- Response status ≠ 200 (ошибка авторизации).
- В 401-ответе указано конкретное поле, которое неверно (утечка информации).
- В системный лог записывается plaintext password (security violation).
- Email с уведомлением о входе НЕ отправляется при успешной авторизации (missing
side effect).
```

Не допускается: «Pass не выполняется». Это не Fail-критерий, это negation.

---

5. Citation conventions

5.1 Inline citation

Каждое утверждение в BR/SR/SPEC имеет inline-citation в квадратных скобках:

> *При первом запуске отображается экран онбординга. [ADAPT-001 §14.1 Forward]*

Формат: `[<id> <section>]` или `[<id> <section> line <N>]` для точной строки.

5.2 Derived маркер

Если утверждение не цитата из ADAPT, а логически выведено:

> *Кнопка `Назад` блокируется на экране онбординга. [ADAPT-001 §14.1 Forward, derived: ADAPT требует «нельзя пропустить онбординг» → блокировка системного `back`]*

Объяснение `derived` обязательно.

5.3 Multi-source

Если утверждение поддерживается несколькими источниками:

> *Пароль хранится в зашифрованном виде. [ADAPT-001 §4 НФТ-002, ISO 27001 A.10.1.1]*

5.4 Что НЕ citation

- Ссылка на сам код («см. `auth/login.py`») — не источник требования.
- Ссылка на тикет в багтрекере — не источник требования (см. 01-glossary.md §1 Цепочка авторитетности).
- Ссылка на чат / устный разговор без записи в ADAPT backward → asked-to-client → answered.

---

6. System prompt для AI-генератора

Каждый AI-агент, генерирующий RENAR-артефакт, получает system prompt из:

1. **Role:** «Ты архитектор требований по стандарту RENAR».
2. **Style guide reference:** ссылка на этот документ.
3. **Glossary reference:** ссылка на 01-glossary.md.
4. **Constraints:**
   - Каждое утверждение либо с citation, либо `derived` с объяснением.
   - Использовать только канонические термины.
   - Структура и длина — согласно §2 этого документа.
   - Никаких запрещённых слов из §4.2.
5. **Output schema:** точный YAML frontmatter format (см. 02-schemas.md).
6. **Examples:** 2-3 примера good/bad для каждого типа артефакта.

Prompt-templates хранятся в `prompts/` директории организации с версионированием:

- `prompts/adapt-from-tz.md@v2.1`
- `prompts/decompose-adapt.md@v2.1`
- `prompts/generate-tc-pos-neg.md@v1.0`
- `prompts/critic-review-sr.md@v1.2`

`ai-provenance.prompt-template` во frontmatter артефакта содержит точную версию.

---

7. Стиль для разных моделей

Разные LLM имеют разные tendencies. Style guide учитывает это через prompt constraints:

Модель	Tendency	Mitigation

Claude Opus	Verbose, склонен к hedging («может быть», «возможно»)	Explicit constraint в prompt: «без hedging modal verbs».
Claude Sonnet	Конкретный, может пропустить edge case	Состязательный критик catches gaps.
GPT-4 / o-series	Marketing language tendency	Explicit blacklist слов из §4.2.
Mini / Haiku	Hallucinated citations	Citation validator hook (см. AIR-07).
Gemini Pro	Иногда смешивает RU/EN	Lang constraint в prompt + post-validation.

Руководство по стилю не запрещает использование любой модели, но требует обеспечения соблюдения через пост-генерационную валидацию.

8. Validation pipeline

После генерации RENAR-артефакта запускается автоматическая валидация:

```
AI generates artifact
    ↓
Style validator (отдельный AI с другим prompt или rule-based)
    ├── citation check (each assertion has [...] или derived marker)
    ├── lexicon check (no forbidden words from §4.2)
    ├── structure check (required sections present)
    ├── length check (within bounds from §2)
    ├── canonical terms check (no forbidden synonyms)
    ├── modal verb check (нет hedging)
    ↓
On fail: regenerate с refined prompt или human review.
On pass: enter draft → ready transition (далее QG-1 / состязательный обзор / утверждение).
```

Валидаторные хуки — нативные для носителя. На git — pre-commit; на document store — pre-save document validator; на любом другом — эквивалентный гейт.

9. Stylistic decisions для RU/EN

9.1 Body language

Body артефакта пишется на языке проекта:

- Российский клиент → RU.
- Международный → EN.
- Двуязычный → primary lang в `lang:` поле frontmatter; second lang — отдельный artifact с `replaces` / `replaced-by` links или translations подпапка.

9.2 frontmatter всегда канонический

Frontmatter поля — всегда канонические (английская латиница):

- `type: BR` (не `тип: БТ`).
- `status: approved` (не `статус: утверждено`).
- `priority: must` (не `приоритет: обязательно`).

UI отображает русские переводы (см. 01-glossary.md §4.5). Frontmatter — машинный, не для UI.

9.3 Mixed-lang запрещён

В пределах одного утверждения mixing RU+EN недопустим:

Плохо:

> *Система должна возвращать `JWT token` после successful login.*

Хорошо (RU):

> *Система должна возвращать JWT-токен после успешной аутентификации.*

Хорошо (EN):

> *The system must return a JWT token after successful authentication.*

Технические термины (`JWT`, `OAuth`, `gRPC`, `RBAC`) — допустимы в любом языке без перевода.

---

10. Перекрёстные ссылки

- Закрытый список канонических терминов и запрещённых синонимов — 01-glossary.md.
- Формальные frontmatter schemas — 02-schemas.md.
- AI-риски и mitigations — 03-ai-risk-register.md.
- Knowledge graph schema (используется validator для cross-reference checks) — 05-knowledge-graph-schema.md.

AI Style Guide RENAR 1.0-draft — renar.tech

Схема графа знаний

> ***Назначение:*** *формальная схема графа знаний (KG) для RENAR-проектов — узлы, грани, properties, query patterns. KG — **derived view** от RENAR-артефактов для семантических запросов AI-агентов и reconciliation. Machine-readable edge types — §3; нормативные поля связей — standard/06 §6.10, standard/08.*

KG — **не источник правды**. Источник истины — артефакты (ADAPT / BR / SR / SPEC / TC) на носителе. Граф derived from frontmatter и не редактируется напрямую. Если граф противоречит артефактам → rebuild графа, не правка артефактов.

1. Use cases

Без KG AI-агент имеет плоский поиск (FTS5/RAG + `parent` / `children` direct hops + keyword grep) — синтаксический контекст. Граф добавляет семантический:

Запрос	Без графа	С графом
Все BR, влияющие на Sales Cycle KPI	Grep + парсить frontmatter	Один Cypher-запрос
При изменении SR-05 — какие задачи и SPEC затронуты	Сканировать все ссылки	Single graph traversal
Какие SPEC-AI требуют high-risk classification по AI Act	Вручную	One query
Карта заинтересованных сторон для проекта	Несколько запросов	Single subgraph extraction
Cross-project dependencies через SPEC-INT	Federation API calls	Federated graph query
Стало ли требование SR-12 деградировать	Manual analysis	Trend analytic on node properties
Stale TC (verifies SR с обновлённой версией, last-run на старой)	Manual check	One query

2. Node types (закрытый список)

Type	Источник	Identity
`Requirement`	BR / SR / TR артефакты	`<id>`
`Specification`	SPEC-* артефакты (9 типов)	`<spec-id>`
`ADAPT`	ADAPT-NNN артефакты	`<adapt-id>`
`BackwardFinding`	B-NNN записи внутри ADAPT	`<adapt-id>:B-NNN`

`TestCase`	TC артефакты (вкл. `tc-type: contract`)	`<tc-id>`
`WorkOrder`	T3 и delta-T3	`<tz-id>`
`Stakeholder`	frontmatter `business-context.stakeholder`	`<stakeholder-id>`
`BusinessGoal`	frontmatter `business-context.business-goal` (deduplicated)	`<goal-id>`
`KPI`	frontmatter `business-outcome.kpi-name`	`<kpi-id>`
`Task`	TR / runtime task store	`<task-id>`
`CodeArtifact`	TC `automation.location`, code commits	`<repo>:<path>:<symbol>`
`Decision`	Architectural decision records	`<decision-id>`
`DeadEnd`	Failed approaches (опционально)	`<deadend-id>`
`RiskItem`	AI Risk Register entry	`AIR-NN`
`Compliance`	Compliance standard reference	`<std>:<control>`
`Template`	Requirements library template	`<template-id>`

Список закрыт; новые типы узлов — только через изменение полного RENAR Standard.

3. Edge types (закрытый список)

3.1 Иерархия и derivation

Edge	From	To	Семантика
`parent`	Requirement	Requirement	A.parent = B → B is parent of A (BR → SR → TR)
`derived-from-adapt`	Requirement / Specification	ADAPT	Артефакт выведен из approved ADAPT section
`derived-from-template`	Requirement / Specification	Template	Шаблон (с version pin)
`replaces`	Requirement / Specification	Requirement / Specification	new replaces old (deprecated)
`supersedes`	Requirement / Specification	Requirement / Specification	strictly newer version (post-delta)
`parent-adapt`	ADAPT	ADAPT	delta-ADAPT → main ADAPT

3.2 ADAPT specifics

Edge	From	To	Семантика
`from-tz`	ADAPT	WorkOrder	source-tz pointer
`parent-adapt`	ADAPT	ADAPT	delta-ADAPT → корневой (`parent-adapt`)
`supersedes`	ADAPT	ADAPT	дезавуирующий ADAPT → дезавуируемый; обратное `superseded-by` (standard/07 §7.6.4); цель переходит в `superseded`
`contains-backward`	ADAPT	BackwardFinding	B-NNN запись
`backward-asks-stakeholder`	BackwardFinding	Stakeholder	who answers
`resolved-by-answer`	BackwardFinding	(timestamp + author)	client-answer record

3.3 SPEC graph (parallel axis)

Edge	From	To	Семантика
`constrained-by`	Requirement	Specification	SR constrained by SPEC-* (typed edge)
`implements-spec`	Task	Specification	TR implements SPEC-*
`depends-on`	Specification	Specification	SPEC depends on another SPEC (DAG)
`referenced-by`	Specification	Requirement / Task	inverse (auto-derived)

3.4 Verification и реализация

Edge	From	To	Семантика
`verifies`	TestCase	Requirement / Specification	TC verifies artifact
`verified-by`	Requirement / Specification	TestCase	inverse (auto-derived)
`implements`	Task	Requirement	Task implements requirement
`realises-in`	TestCase	CodeArtifact	TC.automation.location
`linked-defect`	Requirement	Task (defect type)	Bug на этом требовании

3.5 Происхождение

Edge	From	To	Семантика

`from-order`	ADAPT / Requirement	WorkOrder	source.tz-section reference
`delta-from`	WorkOrder	WorkOrder	delta-T3 → base T3
`cited-in`	Requirement / Specification	WorkOrder	inline citation pointer на раздел T3
`decided`	Requirement / Specification	Decision	Decision при декомпозиции
`deadend`	Requirement / Specification	DeadEnd	Failed approach

3.6 Business / governance

Edge	From	To	Семантика
`owned-by`	Requirement	Stakeholder	business-context.stakeholder
`goal`	Requirement	BusinessGoal	business-context.business-goal
`impacts-kpi`	BusinessGoal	KPI	KPI driven by goal
`compliance-with`	Requirement / Specification	Compliance	compliance entry
`risk-mitigates`	Requirement / Specification	RiskItem	mitigates AIR-NN
`risk-introduces`	Requirement / Specification	RiskItem	introduces new risk (warning trigger)

3.7 Cross-project / integration

Edge	From	To	Семантика
`participates-in`	Specification (SPEC-INT)	Specification (SPEC-INT)	SPEC-INT participants — стороны интеграционного контракта
`cross-deps-on`	Task (project A)	Task (project B)	Cross-project dependency
`integrates-via`	Requirement	Specification (SPEC-INT)	Через SPEC-INT contract

3.8 Edge properties

Edges имеют properties (Cypher-style):

```
(SR:Requirement)-[v:verifies {requirement-version: "1.2", confidence: "high"}]->
(TC:TestCase)
(BR:Requirement)-[c:compliance-with {control: "A.5.34", rationale: "PII
protection"}]->(GDPR:Compliance)
(WO:WorkOrder)-[d:delta-from {effective-date: "2026-05-15"}]->(WO-prev:WorkOrder)
(SR:Requirement)-[cb:constrained-by {since-version: "1.0"}]->(SPEC:Specification)
```

4. Cypher-style query examples

4.4 Stale TC (criteria-version drift)

```
MATCH (r:Requirement)<-[:verifies]-(tc:TestCase)
WHERE tc.last-run.requirement-version < r.version
RETURN r.id, tc.id, tc.last-run.requirement-version, r.version
```

4.5 Multi-hop: code → test → SR → BR → KPI

```
MATCH (code:CodeArtifact {path:"src/auth/registration.py"})
      <-[:realises-in]-(tc:TestCase)
      -[:verifies]->(sr:Requirement {type:"SR"})
      -[:parent*1..2]->(br:Requirement {type:"BR"})
      -[:goal]->(g:BusinessGoal)
      -[:impacts-kpi]->(k:KPI)
RETURN code.path, sr.id, br.id, g.name, k.name
```

«Какие KPI зависят от этого файла кода?» — за один query.

4.6 SPEC dependency cycle detection

```
MATCH path=(s1:Specification)-[:depends-on*]->(s1)
RETURN path
```

Returning rows → нарушение DAG invariant (02-schemas.md §9).

4.7 ADAPT с open backward findings

```
MATCH (a:ADAPT {status:"draft"})-[:contains-backward]->(b:BackwardFinding
{status:"open"})
RETURN a.id, count(b) as open_count
ORDER BY open_count DESC
```

> ***Примечание о нумерации.*** *Используются §4.4-§4.7 для сохранения cross-refs из 02-schemas.md §9 на конкретные queries (Stale TC, SPEC cycle, ADAPT open). Дополнительные queries (BR→KPI, SR-affected tasks, PII→GDPR scope) — derived из patterns §4.4-§4.7 + node/edge таблиц §2-§3.*

5. Derivation rules

KG — derived от frontmatter артефактов. «Прямого редактирования графа» не существует.

Node type	Источник в frontmatter		Edge	Источник
`Requirement`	`id`, `type ∈ {BR,SR,TR}`		`parent`	`parent.id` field
`Specification`	`id`, `type ∈ {SPEC-ARCH..SPEC-OPS}`		`verifies`	`verifies[].id` в TC
`ADAPT`	`id`, `type: ADAPT`		`constrained-by`	`constrained-by[]` в SR
`BackwardFinding`	ADAPT body parsing		`depends-on`	`depends-on[]` в SPEC
`TestCase`	`id`, `type: TC`; derived: `last-run.*`, `last_modified` (§7 SQLite schema), `criteria_changed_at` (§6.5)		`implements-spec`	`implements-spec[]` в TR
`WorkOrder`	TZ-NNN frontmatter		`owned-by`	`business-context.stakeholder` → Stakeholder
`Stakeholder` / `BusinessGoal` / `KPI`	deduplicated из `business-context.*` / `business-outcome.kpi-name`		`compliance-with`	`compliance[]` массив
`Compliance`	`compliance.standard` + `compliance.control`		`derived-from-adapt` / `from-order` / `delta-from`	`source.adapt` / `source.tz-section` / `parent-adapt`

Refresh policy. Граф **rebuilt** при изменении в `.req` репозитории / collection (post-commit/post-save hook), import TC last-run от CI бота, создании/обновлении task. Rebuild **incremental** (только затронутые узлы и грани); full rebuild — раз в неделю reconciliation-агентом.

6. Validation queries (reconciliation)

6.1 Orphan approved requirements

```
MATCH (r:Requirement {status:"approved"})
WHERE NOT (r)-[:verified-by]->()
RETURN r.id  // approved без TC — warning
```

6.3 Circular parent chain

```
MATCH path=(r1:Requirement)-[:parent*]->(r1)
RETURN path
```

6.5 Test fitting suspicious (AIR-06 signal)

```
MATCH (tc:TestCase)
WHERE tc.last_modified < tc.criteria_changed_at
  AND tc.last-run.result = "pass"
RETURN tc.id  // criteria недавно меняли, тут же passing — подозрительно
```

> *Свойства узла `TestCase`: `last_modified` — из node-таблицы (см. §7 SQLite schema); `criteria_changed_at` — derived: timestamp последнего change-record с маркером `[test-spec-change]` (01-glossary.md §2.12). Оба заполняются при derivation графа (§5).*

6.6 SR без constrained-by (missing SPEC links)

```
MATCH (sr:Requirement {type:"SR", status:"approved"})
WHERE NOT (sr)-[:constrained-by]->(:Specification)
RETURN sr.id  // SR approved, но не привязан ни к одному SPEC — warning
```

> ***Дополнительные validation queries*** *(broken citations, orphan stakeholders, orphan SPEC) — derived patterns из §6.1 + §6.5 + node/edge таблиц §2-§3.*

---

7. Нативные для носителя реализации

Граф derived от frontmatter всех `.md` в `<project>.req/` + task store. Рекомендуемая реализация для git-носителя — embedded SQLite с таблицами `nodes(id, type, properties JSON, last_modified)` и `edges(from_id, to_id, edge_type, properties JSON)` с индексами по `from_id`, `to_id`, `edge_type`. Альтернатива для больших проектов: embedded graph DB (Kuzu, RedisGraph local).

Документные носители используют native graph queries через design views / Cypher-like языки — separate infrastructure не требуется.

Schema invariants (независимо от носителя): Типы узлов и типы граней — фиксированы (закрытый список). Properties могут эволюционировать (minor schema bump). Удаление типа узла/грани — major schema bump + migration.

---

8. Federated queries (cross-project)

Координация нескольких проектов через KG federation. Пример: «какие cross-team integrations имеют version drift».

```
MATCH (s1:Specification {project:"auth", type:"SPEC-INT"})
      -[:participates-in]->(int:Specification)
      <-[:participates-in]-(s2:Specification {project:"billing"})
WHERE int.contract-version <> s1.implemented-version
RETURN s1.id, s2.id, int.id, int.contract-version, s1.implemented-version
```

Federation — зависимая от носителя операция; реализуется через convention (межносительный query layer) или native multi-tenant graph.

9. Implementation roadmap

Уровень зрелости	Покрытие графа
RENAR-1 / Core	KG опционален; парсинг frontmatter достаточно.
RENAR-2 / Foundation	Базовый граф: Requirement, TestCase, WorkOrder, Task; edges parent, verifies, verified-by, implements. Простые pre-built queries.
RENAR-3 / Team	+ SPEC, ADAPT, BackwardFinding, Stakeholder, BusinessGoal, KPI, Decision. Edges: constrained-by, depends-on, derived-from-adapt, owned-by, goal, impacts-kpi. AI prompts с graph context.
RENAR-4 / Enterprise	Полная схема + reconciliation queries + federation. Visualization в UI.
RENAR-5	Trend analytics, межносительная federation, embedded graph DB для больших репо.

10. Перекрёстные ссылки

- Каноничные termini узлов и граней — 01-glossary.md.
- Формальные frontmatter schemas (источник derivation) — 02-schemas.md.
- AI Risk Register (AIR-10 KG poisoning) — 03-ai-risk-register.md.
- Style guide (validator использует KG для cross-reference checks) — 04-ai-style-guide.md.

Knowledge Graph Schema RENAR 1.0-draft — renar.tech

RENAR RU Style Guide

> ***Нормативный артефакт.*** *Этот документ — canonical Style Guide для RU нормативного корпуса RENAR Standard: терминология (§1), RFC-2119 RU wording (§2), форматирование (§3).*
>
> ***Применение:*** *любая editorial правка RU контента в* `standard/` , `reference/` , `core/` *(и с soft-application —* `guide/` *).*
>
> ***Operational enforcement:*** `scripts/style-guide-check.js` *+ сопутствующие гейты (* `scripts/check-rfc-modals.js` , `scripts/check-substrate-term.js` , `scripts/check-literary-headings.js` *). Style Guide задаёт* ***правила****, скрипты —* ***enforcement****.*
>
> ***История версий:*** *CHANGELOG.md.*

§0 Introduction

§0.1 Назначение

Style Guide фиксирует **форму** RU нормативного контента:

- какие термины остаются латиницей, какие переводятся в prose, какие — кальки (§1);
- regime для RFC-2119 keywords (RU lowercase canonical, явный carve-out per RFC 8174) (§2);
- canonical форматирование citations / code-fences / headings / tables / typography / frontmatter (§3).

Style Guide **не** dictates per-sentence wording — задаёт constraints, внутри которых editor применяет judgement.

§0.2 Normative status

Style Guide — **normative companion** RENAR Standard.

- Любая editorial правка RU нормативного контента **обязана** соответствовать §1–§3.
- При расхождении формулировок — побеждает `standard/04-terms.md` (canonical terminology); §1 этого Style Guide отражает результат.

§0.3 Scope

In scope: RU prose, headings, tables, lists, code-fences, frontmatter в `standard/` , `reference/` , `core/` , `guide/` ; cross-file links; нормативная семантика (no MUST → SHOULD downgrade); RU typography.

Out of scope: EN-переводы (`*/en/` — отдельный EN Style Guide, deferred); код в `scripts/` , `site/` , `.tausik/` (governed by tech-stack conventions); imagery / SVG (deferred); `research/` drafts pre-publish (author discretion).

§0.4 Hierarchy of authority

При конфликте источников побеждает первый match:

1. `standard/04-terms.md` — canonical terminology RENAR.
2. `standard/01-scope.md §1.7` — closed-list policy.
3. Style Guide §1–§3 — operational normative form.
4. SENAR (родительский стандарт) — для общеинженерной терминологии.
5. ISO/IEC/IEEE 29148:2018 — для requirements engineering терминов.

§0.5 Change procedure

Изменения Style Guide — formal procedure (mirrors closed-list policy `standard/01 §1.7`):

1. Propose — `research/ru-style-amendment-NNN-<topic>.md` с rationale.
2. Review — Style Guide editor (или project owner).
3. Pilot — accepted amendment проходит pilot validation на 1 главе.
4. Lock-in — merged + повышение версии.
5. Retroactive — уже пройденные главы пересканируются на compliance.

§0.6 Versioning

Version	Status	Дата
v1.0	approved	2026-05-17 (Phase 3 pilot validated)
v1.1.x	maintenance	2026-05-19..20 (patches)
v1.2	**approved**	**2026-05-27 (compression pass — historical content в CHANGELOG.md)**

§1 Терминология

§1.1 Принципы

Шесть нормативных принципов, обязательные для применения Style Guide:

1. **No mass find-replace** — каждое term-decision контекстуальное. Слепая поиск-замена ломает нормативный смысл. Editor работает per-occurrence.
2. **Preserved normative semantic** — правка, меняющая RFC-2119 level (MUST/SHOULD/MAY discrimination) — это content change, не editorial. См. §2.5.
3. **AI-bias guard** — LLM trained on web English вписывает anglicism («имплементация», «эссенциальный») и tolerate их в editorial review. Применяется explicit checklist + human spot-check, не «звучит нормально».
4. **Closed list canonical terms** — §1.9 фиксирует closed list canonical RENAR terms (mirrors `standard/04`). Локальное создание новых на уровне проекта — запрещено.
5. **Single source of truth** — canonical normative источник: `standard/04-terms.md`. `01-glossary.md` — informative companion.

6. **Domain-context preservation** — технические термины носителя / VCS / SE (`commit` , `merge` , `hook` , `pipeline` , `runner` , `linter`) остаются латиницей: они конвенции отрасли, RU-эквиваленты теряют точность.

§1.2 Шесть buckets

Все термины RU нормативного корпуса классифицируются в одну из шести buckets:

Bucket	Категория	Policy	Section
A	Technical identifiers (носитель / VCS / SE domain)	Keep latin	§1.3
B	Status vocabulary (lifecycle статусы)	Keep latin (matches YAML schema)	§1.4
C	Organisational vocabulary	Rewrite в RU (prose); keep в YAML/code	§1.5
D	Accepted кальки	Keep, density-cap 1% per chapter	§1.6
E	Rewrite кальки	Per-term replacement table	§1.7
F	Translit	Triage: accepted / borderline / rewrite	§1.8

§1.3 Bucket A — technical identifiers (keep latin)

Правило: остаются латиницей в prose, code-fence content, frontmatter, section labels (когда technical).

Term	Domain	RU UI projection (только в `guide/`)
`commit`	VCS	фиксация
`merge` / `diff` / `branch` / `push` / `pull`	VCS	слияние / различие / ветка / отправка / получение
`hook` / `patch`	VCS	перехватчик / патч
`frontmatter`	Markdown	заголовочный блок
`manifest`	RENAR	манифест
`slug` / `hash`	Markdown / crypto	слаг / хеш
`trigger`	RENAR enforcement	триггер
`runner`	CI / testing	прогонщик
`pipeline`	CI	конвейер
`release` / `deploy` / `build`	DevOps	релиз / развёртывание / сборка
`workflow`	RENAR / VCS	рабочий поток
`linter` / `parser` / `compiler`	tooling	линтер / парсер / компилятор

`tracker`	RENAR	трекер
`substrate` (только поля/код/имена файлов)	RENAR	**носитель** (prose canonical)
`provenance`	RENAR	происхождение
`fallback`	SE / RENAR ops	запасной вариант

RU UI projection — допустима в multilingual UI (`standard/04 §4.13.3`); в `standard/` , `reference/` , `core/` — всегда canonical латиница.

> ***`substrate`* → «носитель» в prose.** *Канонический термин в RU prose — **«носитель»** (склоняется). Латиницей `substrate` остаётся только в именах полей (`substrate-capabilities`), коде/YAML и именах файлов (`03-substrate-versioning.md`). Канонический источник — `standard/04 §4.8` .*

Cross-bucket extension: Bucket A whitelist открытый — новые VCS / SE / CI / observability domain-terms добавляются proactively при необходимости. Brand names (`GitHub` , `Astro` , `Markdown`) — proper nouns, не Bucket A.

§1.4 Bucket B — status vocabulary (keep latin)

Правило: lifecycle status identifiers (`draft` , `proposed` , `approved` , `verified` , `accepted` , `done` , `obsolete` , `deprecated` , `frozen` , `active` , `planning` , `blocked` , `review` , `ready` , `passing` , `failing` , `client-ready` , `answered` , `resolved` , `revised`) остаются латиницей в prose, frontmatter, code-fences.

Rationale: status — system-readable identifier, появляется в YAML (`status: approved`); state machines `standard/10` определены через эти identifiers; перевод в prose разрывает референциальную цепь.

RU UI projection (допустима только в `guide/`): `draft` → «черновик», `approved` → «утверждённый», `verified` → «проверенный», `accepted` → «принятый», `done` → «выполненный», `deprecated` → «удалённый», `frozen` → «замороженный», и т.д. См. полный список в `01-glossary.md §2.6` .

Case consistency: всегда lowercase. `Proposed` / `ПРЕДЛОЖЕННЫЙ` — paste-error.

§1.5 Bucket C — organisational vocabulary (rewrite в prose)

Правило: в **prose** заменить на RU; в **YAML field names, code, fence content** — keep latin.

Latin (prose)	RU canonical	Notes
`scope`	область (применения) / охват	YAML key `scope:` keeps latin
`audit`	ревизия / проверка / контроль	прозовый `audit trail` / `audit log` → «журнал аудита» (§1.14); идентификатор `audit-trail` / путь — keep latin

`policy`	правило / политика	«closed list policy» — keep latin (RENAR canonical phrase)
`ownership`	владение / ответственность	—
`default`	по умолчанию	YAML `default:` keeps latin
`bypass` / `rollback` / `override` / `stub`	обход / откат / переопределение / заглушка	—
`guide` (noun)	руководство	директория `guide/` — keep latin (path)
`walkthrough`	прохождение / пошаговый разбор	—
`quickstart`	быстрый старт	—

Context discrimination: перед rewrite — проверить:

- Pure prose («scope нормирует X») → replace.
- YAML / code field name (`scope:`) → keep.
- Идентификатор / путь домена носителя (`audit-trail` , `commit`) → keep latin; прозовый `audit trail` / `audit log` → «журнал аудита» (§1.14).
- Cross-ref label («closed list policy» как RENAR canonical phrase) → keep latin.

§1.6 Bucket D — accepted кальки (keep, density-cap 1%)

Правило: accepted кальки остаются как есть; density-cap **1% prose-words per chapter**. При превышении — реструктуризация / synonyms / pronominals.

Accepted list: `нормативный` , `реализация` , `спецификация` , `формальный` , `опциональный` , `атомарный` , `декомпозиция` , `композиция` , `верификация` , `валидация` , `идентификация` , `идентификатор` , `интеграция` , `миграция` , `авторизация` , `аутентификация` , `итерация` .

Density measurement: density = (term occurrences) / (prose words after strip YAML/fences/inline-code) × 100. Cap: 1.0% per chapter. Enforcement — `scripts/style-guide-check.js` density mode.

§1.7 Bucket E — rewrite кальки

Правило: заменить на RU canonical per replacement table. Manual per-occurrence (no mass-replace per §1.1).

Latin (calque)	RU canonical	Notes
`имплементация`	реализация (общее) / исполнение (active-voice)	—
`конформация`	соответствие	**Exception:** `conformance` — ISO 29148 term, keep latin в контексте носителя (`standard/13`)

`эссенциальный`	существенный / обязательный	context-dependent
`дрифт` (translit)	дрейф / расхождение	**Exception:** `drift` (latin) — RENAR canonical class name; keep `Schema drift / Lifecycle drift` латиницей
`трейс*` (in prose)	трасс* / отслеживание	`traceability` — RENAR canonical, keep latin
`аннотация` (AI-bias overuse)	пояснение / примечание	context-dependent
`регуляция`	регулирование	rare, AI-bias residue

Exceptions: frozen quote / citation — keep verbatim; field name в YAML schema — never touch; cross-ref label как canonical phrase — keep latin per §1.3.

§1.8 Bucket F — translit triage

Транслит-термины классифицируются:

- **Accepted (keep):** `артефакт`, `автор`, `шаблон`, `репозиторий` — давно accepted в RU IT.
- **Borderline (case-by-case):** `ревью` (keep после «adversarial»; rewrite stand-alone → «рецензия»), `кейс` (keep в «edge case» / «test case»; rewrite общий → «случай»), `стейкхолдер` (keep formal; «заинтересованная сторона» longer-formal для `standard/`), `чекпойнт`, `гейт`, `онбординг`.
- **Rewrite (low-count):** `фронтенд` / `бэкенд` → keep latin form (`frontend` / `backend`); `пайплайн` → keep `pipeline`; `фолбэк` → keep `fallback`; `оверхед` → «накладные расходы»; `апдейт` → «обновление».

Negative scenario: не делать translit того, что Bucket A keeps latin. `pipeline` (latin) > `пайплайн` (translit).

§1.9 Closed list canonical RENAR terms

Канонический список — `standard/04-terms.md §4.3–§4.7` (артефакты, SPEC family, TC types, Quality Gates, lifecycle статусы, V1–V6 capabilities, drift classes). Style Guide отражает этот canonical; не дублирует.

§1.10 Accepted technical loanwords

Cross-bucket whitelist — accepted technical loanwords (не калька, не translit):

Term	Domain
`state machine`	distributed systems
`immutable history`	RENAR / V1 (gloss возможности)
`atomic change unit`	RENAR / V2

`diff & review`	RENAR / V3
`author + timestamp`	RENAR / V6 (gloss возможности)
`adversarial review`	RENAR / AI
`eval` / `evaluation`	AI testing
`runner`	CI / RENAR
`branching`	VCS / V4
`version pin`	RENAR / носитель / V5
`traceability`	RE general
`provenance`	RENAR AI
`closed list`	RENAR meta-policy

Правило: не подлежат rewrite — canonical в RENAR domain и accepted в международной literature.

§1.11 Добавление новых терминов

1. **Identify need** — term появляется в нормативной главе ≥3 раз без §1.9 listing.
2. **Bucket classification** — определить A–F по §1.2 critera.
3. **Rationale draft** — 3–5 предложений: why needed, why this bucket, what alternatives rejected.
4. **Review** — Style Guide editor (или project owner), same procedure как для `standard/04` formal change.
5. **Retroactive** — после утверждения, существующие occurrences нормализуются.

В interim period — допустим в `research/` drafts; в `standard/` / `guide/` / `reference/` / `core/` — только после approval.

§1.12 AI-bias guard — failure modes

LLM trained on multilingual web с English dominance склонна:

- **English-bias** — inserting anglicism («корпоративный admin»), preferring «имплементация» over «реализация», rewriting RU в English-flavored RU.
- **Confidence over precision** — confidently rewriting «должен» → «следует» «для читаемости» (semantic downgrade per §2.5).
- **Pattern-completion drift** — completing 51-й example с тем же pattern, даже если §1 требует RU rewrite.
- **Compression anglicism creep** (failure mode введён v1.3): при сжатии bullet-списков в inline-параграфы LLM сохраняет английские структурные метки (`**Actor:**` , `**Default:**` , `**Override:**`) и одиночные английские слова мид-prose (`actor` , `evidence` , `cadence` , `result` , `denial`). Это **anti-pattern** — после компрессии плотность анг. токенов в prose растёт сверх Bucket A/D, нарушая §1.1 принципы.

Mitigation: Bucket E replacement table (§1.7) — hard-coded substitutions; semantic preservation (§1.1, §2.5) — explicit check; human spot-check на random 10% правок — обязателен (не на «typical» examples — на random); compression-pass §1.13 synonym map (для prose-cleanup сразу при сжатии).

§1.13 Compression synonym map — prose-cleanup при сжатии bullet-списков

При замене bullet-списка / multi-paragraph block на inline-параграф структурные метки и connectives **обязаны** переводиться. Canonical synonym map (closed list для частых compression-failure-modes):

English label (prose)	RU canonical	Контекст применения
`**Actor:**` / `actor`	`**Участник:**` / «участник»	RACI, gates, assessments
`**Methodology:**`	`**Методология:**`	procedures
`**Evidence:**` / `evidence`	`**Доказательная база:**` / «свидетельства»	conformance audit
`**Default:**` / `default` (mid-prose)	`**По умолчанию:**` / «по умолчанию»	cadence, settings
`**Override:**` / `override` (mid-prose)	`**Переопределение:**` / «переопределение»	manifest declared-stricter
`**Cadence:**` / `cadence`	`**Периодичность:**` / «периодичность»	re-assessment, review
`**Result:**` / `result` (mid-prose)	`**Итог:**` / **Результат:** / «итог»	assessment outcome
`**Denial:**` / `denial`	`**Отказ:**` / «отказ»	assessment, gate result
`**Joint:**` / `Joint`	`**Совместный:**` / «совместный»	ownership type
`Single` (в roles table)	«одиночный» / «единоличный»	ownership type
`Split` (в roles table)	«разделённый»	ownership type
`**Recovery plan:**`	`**План восстановления:**`	потеря соответствия
`**Trigger:**` / `trigger` (mid-prose)	`**Триггер:**` (русифицировано, accepted) / «поводом для»	event-driven flows
`**Signal:**` / `signal` (mid-prose)	`**Сигнал:**` / «сигнал»	observable signals
`Review` (как noun mid-prose)	«обзор», «ревью» (если canonical RENAR-термин)	adversarial review остаётся как canonical
`Audit log`	«Журнал аудита» (mid-prose) или `audit-log` (как identifier)	event tracking
`Independent verification`	«Независимая проверка»	third-party assessment
`Notification` (mid-prose)	«Уведомление»	public communications
`Application` (mid-prose, как «Применимость»)	«Применимость»	gate scope

Carve-out (НЕ переводится):

- Backtick-код, YAML-значения, JSON properties — `actor:`, `result: pass`, `last-run:`, `default: ...` остаются как есть (это API / schema fields).
- Закрытые RENAR-термины (§1.9): `manifest`, `ADAPT`, `BR`, `SR`, `SPEC-*`, `TC`, `ai-provenance`, `audit-trail`, `declared-stricter`, `assessment-mode`, `mandatory clauses`, `verifies[]`, `conformance` остаются.
- Bilingual H3 headers формата «### §X.Y RU-название (canonical-EN-term)» — допустимы как terminology mapping: «### 13.5.1 Участник (actor)».

Hard rule: новый bold-label paragraph header `**English Word:**` мид-prose, где English Word **не** входит в §1.9 / §1.10 closed lists и **не** имеет canonical RENAR-семантики — нарушение Style Guide v1.3+.

§1.14 Process-vocabulary — нормализация процедурной лексики

Лексика процедуры изменения стандарта и оценки соответствия — **не** canonical-идентификаторы и **не** входит в accepted loanwords (§1.10). Это описательные обороты; латиницей в прозе они нарушают §1.1 (сначала смысл) и §2.2 камертона голоса (один автор, не комитет). Переводятся всегда, грамматически по падежам:

English (prose)	RU canonical	Контекст
`formal change procedure`	формальная процедура изменения (стандарта)	§13.9 — изменение closed lists
`research draft`	исследовательский черновик	этап процедуры изменения
`public review`	публичное обсуждение	этап процедуры изменения
`minor-version bump` / `major-version bump` / `version bump`	повышение minor-/major-версии / повышение версии	выпуск версии стандарта
`migration guidance`	руководство по миграции	сопровождение conformant-проектов
`loss-of-conformance` / `loss of conformance`	потеря соответствия	§13.8
`self-assessment checklist`	чек-лист самооценки	§13.5, reference/08
`corresponding alignment`	согласующая правка	синхронизация с SENAR
`project-local` (adj/adv)	локальный для проекта / локально на уровне проекта	закрытые списки
`audit trail` / `audit log` (прозовый space-form)	журнал аудита	§10.13, §4.8.5; идентификатор `audit-trail` / путь — keep latin
`baseline` (bare prose)	базовый уровень / базовые значения / эталон / опорное состояние (по контексту)	метрики §12; eval/UX §8–9; идентификаторы `baseline-*` / `baselines/*.png` — keep latin

`immutable` (свободное прилаг.)	неизменяемый (по роду/падежу)	V1-gloss `immutable history` (§1.10) — keep; код-фенс комментарии — keep
`approval` (сущ.)	утверждение	имя гейта QG-0 `Approval Gate`, статус `approved`, поля `approval-*`, ISO-поле «Approval authority» — keep
`findings` (проза)	находки	вердикт-метки `no findings` / `findings present`, §7 `backward findings`, метрика `Reconciliation Findings` — keep

Carve-out (НЕ переводится):

- Имена файлов (`08-conformance-self-assessment.md`), HTML-анкеры (`<a id="...self-assessment...">`), URL-фрагменты ссылок — referential integrity.
- Идентификаторы и пути (`loss-event`, `audit-trail/CFM-…/loss-events/…`).
- Билингв-заголовки формата «### RU-название (canonical-EN-term)»: «### 13.5 Процедура самооценки (Self-assessment)» — допустимы как terminology mapping (§1.13 carve-out).
- Версионные квалификаторы `minor` / `major` рядом с «версия» — accepted hybrid (`minor-версия`), переводится только `bump`.

Enforcement: `scripts/check-process-vocab.js` флагует в прозе (вне carve-out) обороты `formal change procedure` … `baseline`; в `check:all`. Три нижних термина (`immutable` / `approval` / `findings`) **не гейтируются** — они переплетены с каноническими именами (QG-0 `Approval Gate`, V1-gloss `immutable history`, метрика `Reconciliation Findings`), поэтому blocklist-гейт даёт ложные срабатывания; прозовые формы нормализуются вручную по этой таблице.

§1.15 Поправка v1.3-draft — расширение русификации прозы

Решением владельца проекта (2026-06) политика русификации прозы расширена: перечисленные ниже термины, ранее остававшиеся латиницей по §1.3 (Bucket A) / §1.7 / §1.10, **переводятся в прозе и табличных лейблах**. В backtick-коде, YAML-полях/значениях, именах файлов, RFC-2119-цитатах и идентификаторах — остаются латиницей. Поправка имеет приоритет над §1.3/§1.7/§1.10 для этих конкретных терминов (§0.4: первый match — но эта таблица фиксирует исход для перечисленного).

Latin (проза)	RU canonical	Latin сохраняется в
`manifest`	манифест; `conformance manifest` → манифест соответствия	имя файла `RENAR-CONFORMANCE.yaml`, поля
`conformance`	соответствие (склоняется); `conformant` → соответствующий; `non-conformant` → несоответствующий	`RENAR-CONFORMANCE.yaml`, `senar-version` / `renar-version`, уровни `RENAR-1..5`
`provenance`	происхождение	поле `ai-provenance`, `*-provenance` в backtick
`adversarial review`	состязательный обзор; `adversarial reviewer` → состязательный рецензент;	поле `adversarial-review-ref`

	`adversarial critic` → состязательный критик	
Имена ролей	`Architect` → Архитектор; `Engineer` → Инженер; `Reviewer` (роль) → Рецензент; `Supervisor` → Супервизор; `Stakeholder` → Заинтересованная сторона; `Product Owner` → Владелец продукта	нормативные имена SENAR §4 в манифесте и значения `role:` (`authorized-role-holder` и т.п.); внешние термины `Stakeholder Requirement` (BABOK), `Product Owner` в цитате Scrum/SAFe

Средний слой (2026-06, решение владельца — «мягкие слова-понятия»): дополнительно переводятся в прозе и лейблах, при сохранении истинных идентификаторов латиницей:

Latin (проза)	RU canonical	Latin сохраняется в
`lifecycle`	жизненный цикл; «lifecycle-состояние/статус» → состояние/статус жизненного цикла	drift-класс `Lifecycle drift`, статусы-значения, имена файлов
`mandatory clauses / mandatory clause`	обязательные положения / обязательное положение; `mandatory` (прил.) → обязательный/обязательно	schema-аннотации `mandatory` в YAML-примерах
`enforcement`	обеспечение соблюдения; `enforcement-точка` → точка контроля	поля/идентификаторы
`SoT inversion / SoT`	инверсия источника истины / источник истины	—
`canonical` (прил.)	канонический	bilingual-заголовки `English (canonical)`, comparison-заголовки `RENAR canonical`
`closed list`	закрытый список	—
`state machine`	машина состояний	§1.10-gloss где как V-описание
`data model` → модель данных; `multilingual` → многоязычный; `spot-check` → выборочная проверка; `normative` → нормативный; `trace chain` → цепочка прослеживаемости; `full-completeness` → полнота		drift-классы (`Order / provenance drift`), внешние термины

Сохраняются латиницей (истинные идентификаторы, §0.7 машиночитаемый слой): **frontmatter**; lifecycle-**статусы** (`draft` / `approved` / `verified` / `frozen` / `superseded` /…) — §1.4 референциальная цепь; имена полей и backtick-код; ID/типы (`BR` / `SR` / `SPEC-*` / `TC` / `ADAPT` / `QG` / `MVR` / `V1–V6`); `Quality Gate(s)` как канон-имя; §1.10 V-glosses

`immutable history` / `atomic change unit` / `diff & review` / `version pin` / `traceability` / `branching` / `author + timestamp` ; Bucket A VCS (`commit` / `hook` / `trigger` / `runner` / `pipeline` /...); карв-аут `backward findings` / `no findings` / `findings present` ; имена собственные и внешние термины (BABOK/ISO/KG node-types).

---

§2 RFC-2119 RU normative wording

§2.1 Принципы

1. **Semantic-fidelity первичен** — RFC-2119 (RFC 2119) фиксирует строгое discrimination между MUST / SHOULD / MAY. RU wording должен сохранить эту дискриминацию.
2. **No semantic downgrade** — Phase 5 editorial pass **не имеет права** менять modal verb level. «должен» → «следует» — это **content change**, требует отдельной задачи.
3. **RFC 8174 carve-out для RU** — RU lowercase modal verbs (`должен` , `следует` , `может` , `рекомендуется` , `допускается` , `требуется` , `обязательно` , `опционально` , `обязан` , `запрещено`) в нормативных clauses RENAR Standard **являются canonical эквивалентом** RFC-2119 UPPERCASE keywords. Их normative weight равен соответствующему UPPERCASE.
4. **No UPPERCASE convention в RU prose** — `ОБЯЗАН / ДОЛЖЕН / СЛЕДУЕТ / МОЖЕТ` — **запрещено**. Capitalization только когда начинают предложение (orthographic). Визуальный emphasis — через `**bold**` .
5. **Imperative-mood для нормативных положений** — нормативные положения используют imperative («X должен Y», «X обязан Y»). Indicative («X выполняет Y», «X is Y») — только для descriptive prose.
6. **EN translation convention** — EN-перевод (`*/en/`) использует UPPERCASE RFC-2119 keywords; RU carve-out на EN **не распространяется**.

§2.2 Regime decision (Option C — RU lowercase canonical)

RENAR Standard RU corpus использует **RU lowercase modal verbs** как canonical normative wording. Решение зафиксировано: de-facto majority уже соответствует; natural RU normative tradition (ГОСТ Р, ISO 29148 RU translations); RFC 8174 carve-out явно устраняет ambiguity.

§2.3 Canonical mapping table

11 RFC-2119 / RFC-8174 keywords ↔ RU canonical equivalents. **Bidirectional usage:** RU pass — EN UPPERCASE → RU lowercase; EN translation — RU lowercase → EN UPPERCASE.

Mandatory levels

RFC-2119	RU canonical (primary)	RU canonical (synonyms)	Semantics
`MUST`	должен	обязан / необходимо / требуется	Absolute requirement
`MUST NOT`	не должен	запрещено / недопустимо	Absolute prohibition

`SHALL`	должен	обязан	Equivalent к MUST per RFC 2119 §2
`SHALL NOT`	не должен	запрещено	Equivalent к MUST NOT
`REQUIRED`	обязателен / обязательно	необходим	Equivalent к MUST (adjective form)

Strong recommendation level

RFC-2119	RU canonical (primary)	RU canonical (synonyms)	Semantics
`SHOULD`	следует	рекомендуется	Strong recommendation; exceptions требуют explicit rationale
`SHOULD NOT`	не следует	не рекомендуется	Strong negative recommendation
`RECOMMENDED`	рекомендуется	предпочтительно	Equivalent к SHOULD
`NOT RECOMMENDED`	не рекомендуется	нежелательно	Equivalent к SHOULD NOT

Optional level

RFC-2119	RU canonical (primary)	RU canonical (synonyms)	Semantics
`MAY`	может	допускается	Truly optional; no preference
`OPTIONAL`	опционален / опционально	необязателен / по выбору	Equivalent к MAY

Negation table — canonical patterns

Form	RU canonical	Discrimination
MUST NOT / SHALL NOT	не должен / запрещено / недопустимо	Hardcoded prohibition
SHOULD NOT / NOT RECOMMENDED	не следует / не рекомендуется / нежелательно	Strong negative recommendation
(MAY NOT — non-canonical RFC)	не требуется	RFC 2119 не содержит MAY NOT; «не требуется» = «MAY omit»

Hard rule: «не должен» (MUST NOT) ≠ «не следует» (SHOULD NOT). Editor никогда не подменяет одну форму другой «для читаемости».

§2.4 Tense / mood normative rules

§2.4.1 Normative clauses → imperative-mood

```
# Canonical
SR должен прослеживаться до родительского BR через `parent:` link.

# Anti-pattern (indicative — теряется normative weight)
SR прослеживается до родительского BR через `parent:` link.
```

§2.4.2 Definitions → predicative form

Canonical form: `X — Y, нормированное Z`.

```
ADAPT — двусторонняя адаптация ТЗ, нормированная процессом согласования с заказчиком.
```

Acceptable variants: `X — это Y, …` (с подчёркивающим частицей «это»); `X — Y` (без «это»).

Anti-pattern: `X является Y` (formal-overhead); `X представляет собой Y` (anglicism-style копи verb chains).

§2.4.3 Future tense — scope-limited

Future-tense (`будет / будут`) **запрещён** в нормативных clauses. Допустим только в: roadmap sections, forward-looking limitations, conditional consequences.

```
# Wrong (semantic ambiguous — это требование или прогноз?)
SR будет содержать parent link.

# Correct
SR должен содержать parent link.
```

§2.4.4 Conditional clauses

Canonical pattern: `Если X, то Y должен Z`. Variants: `При условии X — Y должен Z`; `Когда X, Y обязан Z`.

Anti-pattern: conditional + indicative («Если X, то Y делает Z») — теряется normative weight.

§2.5 Strict semantic preservation rule

Hard rule: Phase 5 editorial pass **не имеет права** менять RFC-2119 level modal verb.

Permitted (semantic-preserving):

From	To	Reason
`должен` ↔ `обязан`	—	Same level (MUST); synonym choice editorial
`следует` ↔ `рекомендуется`	—	Same level (SHOULD)

`может` ↔ `допускается`	—	Same level (MAY)
`необходимо` ↔ `требуется`	—	Same level (MUST); passive-voice register
`не должен` ↔ `запрещено`	—	Same level (MUST NOT); emphatic synonym

Forbidden (content change, не editorial):

From	To	Reason
`должен`	`следует`	MUST → SHOULD downgrade
`следует`	`может`	SHOULD → MAY downgrade
`не должен`	`не следует`	MUST NOT → SHOULD NOT downgrade
Любое modal	indicative («X делает Y»)	Loss of normative weight

Verification protocol: pre-pass scan modal verbs in chapter (count per level); post-pass re-scan; sum-per-level не должна меняться. Если меняется — flag as semantic drift, revert, surface для content-review задачи. Enforcement — `scripts/check-rfc-modals.js`.

§2.6 reference/04 descriptive carve-out

`04-ai-style-guide.md` содержит EN RFC-2119 keywords в **descriptive** context (mentions RFC 2119 vocabulary, не uses normatively). Descriptive mention RFC-2119 keywords в reference / guide / core — **разрешено**, при условии:

1. Контекст явно descriptive: «RFC 2119 определяет `MUST` как absolute requirement».
2. Keyword обёрнут в `code-fence` или blockquote / EN-citation.

Phase 5 chapter pass distinguishes — descriptive mention HE migrates на «должен», keeps EN form.

---

§3 Formatting conventions

§3.1 Принципы

1. **Читаемость носителем первична** — сохранять git-blame readability; не ломать syntax highlighting; быть стабильным при нативной для носителя обработке (markdown lint, frontmatter validator).
2. **Scan-readable вторичен** — numbered sections для cross-ref, consistent bullets, typographic quotes.
3. **No tool dependency** — convention applicable manual editor (любой text editor); не требует specific IDE plugin.
4. **Accessibility** — semantic heading hierarchy (no skip H1 → H3); typed code-fences для syntax-highlighting; alt-text (deferred).
5. **De-facto convention endorsement** — bullet `-` -only (100% базовый уровень), table alignment-less `---` (100% базовый уровень) — endorsed.
6. **RU-типографика первого класса** — `«»`, `—`, NBSP — базовый уровень нормативной RU-типографики.

§3.2 Citations & cross-refs

Три класса:

Class	Pattern	Когда
(A) Bare intra-file	`§3.5`	Reference внутри того же файла
(B) Markdown link cross-file	`[§4.5](../standard/04-terms.md#4.5)`	Reference на другой файл
(C) Anchor-only intra-file	`[§3.5](#35-tables--lists)`	Clickable intra-file reference

Hard rule: cross-file reference **обязан** быть markdown link (Class B). Bare `§X.Y` для cross-file — anti-pattern (не clickable, no PDF outline integration).

Inline «глав X».: «в главе 4» — canonical для cross-file narrative reference. «глава 4» без link — anti-pattern, если cross-file.

«См.» (RU) — endorsed. «see» (EN) — banned.

Anchor format: lowercase, hyphens вместо spaces / dots, RU keywords lowercase Cyrillic. Verified post-edit via cross-ref dead-link scan (`scripts/check-md-links.js`).

§3.3 Code-fences

Hard rule: все fences обязаны иметь language tag. No-lang fences — anti-pattern.

Language tag whitelist (closed list):

Tag	Purpose
`yaml`	frontmatter / config examples
`bash`	CLI commands / shell snippets
`cypher`	Knowledge graph queries (`reference/05`)
`markdown`	Markdown examples внутри docs
`json`	JSON schema / API payloads
`python`	Python script examples
`sql`	SQL queries
`text`	Plain text / tree fragments / generic output

Inline code (backticks): identifiers (`` `SR-001` ``), file paths (`` `standard/04-terms.md` ``), type / field names (`` `parent: BR-001` ``), short CLI fragment.

Anti-pattern: backticks вокруг whole sentences — reserve backticks для technical identifiers.

§3.4 Headings

§3.4.1 Numbering convention — split

Directory	Convention	Rationale
`standard/`	Dotted-number (`## 3.5 Tables / lists`)	Normative — numbered для cross-ref precision
`reference/`	Dotted-number	Same as standard/ — reference is normative companion
`core/`	Plain (`## Что вы получите`)	Pedagogical narrative
`guide/`	Plain или numbered «Phase 0..9»	Narrative + numbered phase walkthroughs
`research/`	Author choice	Drafts

§3.4.2 Depth rule

- Max depth normative — **H3**.
- H4 — exception (substructure внутри large H3), ≤ 5 per chapter.
- H5 / H6 — **banned** (restructure parent instead).
- Depth jumps (H1 → H3 без intermediate H2) — **banned**.

§3.4.3 Casing & form

Headings — sentence-case RU (capitalize only first word + proper nouns). Not title-case English («## Tables And Lists»). **Anti-pattern:** sentence-as-heading («## Этот раздел описывает основные принципы»). Heading — short label, not sentence.

§3.5 Tables / lists

Hard rules:

- Bullet style — `-` only. `*` / `+` — banned. Mixed bullets within file — banned.
- Table alignment — `---` only. `:---` / `---:` / `:---:` — banned (вводят rendering variance per renderer).
- Nested unordered — 2-space indent (CommonMark default).
- Nested ordered — 3-space indent.
- Ordered list — manual `1.`, `2.`, `3.` preferred (explicit count visible в source).

§3.6 frontmatter YAML

Canonical field order:

```yaml
---
title: "Document title"
status: draft  # | proposed | approved | verified | obsolete | frozen
phase: "Phase N description"  # optional
epic: epic-slug  # optional
task: task-slug  # optional
draft-date: 2026-05-16  # ISO 8601
scan-date: 2026-05-16  # для audit artifacts
target-final-path: "reference/06-...md"  # для drafts с known lock-in target
```

```
lang: ru  # primary language
---
```

Quoting policy:

Value type	Quoting
Pure ASCII identifier (`status: draft` , `lang: ru`)	No quotes
String с пробелами / special chars (`title: "..."`)	Double quotes
ISO date (`2026-05-16`)	No quotes
Numeric / Boolean	No quotes

`lang:` **closed list** — `ru` , `en` (planned для EN translations).

Validation: `node scripts/validate-frontmatter.js` (required-field presence + type correctness + closed-value-list).

§3.7 RU typography

§3.7.1 Кавычки — «»

Canonical: ёлочки `«»` для outer quotes. ASCII `"..."` в prose — anti-pattern.

Exceptions: ASCII quotes допустимы внутри code-fences (verbatim), в URL / file paths, в EN citation внутри RU prose.

§3.7.2 Em-dash — `—`

Em-dash `—` (U+2014) для: парных separators («Term X — described as Y — applies …»), definition predicate («X — Y»), list-item separator («item A — description»).

Anti-pattern: `-` (hyphen) или `--` (double-hyphen) instead of `—` .

Exceptions: hyphen `-` correct в compound words («closed-list», «source-of-truth»), file paths, identifiers. En-dash `–` (U+2013) для number ranges («§3.5–3.7») — rare, acceptable.

§3.7.3 Non-breaking space

Required:

- Между числом и единицей: `100 ms` .
- Между initials и фамилией: «И. М. Сеченов».
- Перед em-dash в pair: «слово — описание».

Markdown source может содержать literal NBSP (U+00A0) или HTML entity ` ` . Soft rule — author discretion.

§3.7.4 Числительные форматы

Form	Convention
Whole numbers ≤ 999	No separator: `42` , `999`

Whole numbers ≥ 1000	Space separator: `1 000`, `50 649`
Percentages	`33%` (no space)
Decimals	English period `0.5` (corpus default)
Ordinals	`1-й`, `2-й` (RU suffix)

§3.7.5 Math operators

`≥` / `≤` / `≠` / `±` / `→` / `↔` — Unicode preferred в prose. ASCII `>=` / `<=` / `!=` / `+/-` / `->` — anti-pattern в prose; acceptable inside code-fences.

§3.8 Cross-file link conventions

- **Relative paths only** (from linking file's directory). Absolute paths — break при site build.
- **Anchor case:** lowercase, hyphens, RU keywords Cyrillic.
- **Dead-link prevention:** Astro build / `scripts/check-md-links.js` flag broken links.

```
# from reference/06-ru-style-guide.md
[§4.5 standard/04](../standard/04-terms.md#4.5)
[reference sibling](01-glossary.md)
```

§3.9 Inline emphasis (bold / italic)

Bold (`**text**`) — для term introduction («**Closed list policy** — формальный механизм ...»), emphatic normative («**Hard rule:** ...»), negative scenario flag («**Anti-pattern:** ...»).

Hard rule: bold **не используется** как replacement для UPPERCASE (§2.1, принцип 4). Bold — additive emphasis, не shouting.

Italic (`*text*`) — restricted: foreign-language phrase («*ad hoc*», «*de facto*»), title of external work («*Software Requirements* (Wiegers)»). Не для general emphasis (use bold).

---

§4 Integration & enforcement

§4.1 Canonical sources

Источник	Назначение
`standard/04-terms.md`	Canonical RENAR terminology (closed list canonical из §1.9)
`standard/01-scope.md §1.7`	Master index 16 closed lists
`01-glossary.md`	Informative glossary с примерами и mapping на ISO/BABOK/SAFe
`02-schemas.md`	Canonical frontmatter schemas

§4.2 Automated enforcement (scripts/)

Style Guide задаёт правила; следующие скрипты enforce их:

Скрипт	Что проверяет	Style Guide §
`scripts/validate-frontmatter.js`	required fields, type correctness, closed value lists	§3.6
`scripts/check-rfc-modals.js`	EN UPPERCASE / RU UPPERCASE modals; modal-verb level integrity	§2.1, §2.5
`scripts/check-substrate-term.js`	`substrate` в prose → «носитель»	§1.3
`scripts/check-substrate-leakage.js`	git-specifics в normative-каталогах	§1.3
`scripts/check-literary-headings.js`	sentence-as-heading anti-pattern	§3.4
`scripts/check-md-links.js`	dead links, anchor validity	§3.2, §3.8
`scripts/check-site-russian-prose.js`	site-build RU prose checks	§1, §2
`scripts/style-guide-check.js`	aggregate Style Guide enforcement	§1, §2, §3

npm scripts: `npm run check:all` запускает полный gate sweep.

§4.3 Change procedure (cross-ref)

См. §0.5 Change procedure.

§5 Limitations & follow-ups

- **EN Style Guide** — отложен до начала EN translation epic. Будет mirror этого RU Style Guide с swapped polarity (UPPERCASE EN canonical, lowercase RU только в citation context).
- **Imagery / SVG alt-text policy** — отложено до введения images в корпус.
- **Inner quotes** (`„…"` — German low-9 + high-9) — defer to author discretion case-by-case; формального правила пока нет.
- **NBSP insertion** — soft rule; не gated check. Возможен дальнейший lock-in.

§6 Источники

- **RENAR Standard** — `standard/`.
- **SENAR (родительский стандарт)** — методологическая база.
- **RFC 2119** — Key words for use in RFCs (1997).
- **RFC 8174** — Ambiguity of Uppercase vs Lowercase (2017).
- **ISO/IEC/IEEE 29148:2018** — Requirements engineering.

- **ГОСТ Р, ISO/IEC** в RU translations — natural RU normative tradition (carve-out per §2.1, RFC 8174).

RENAR RU Style Guide v1.2 — compression pass 2026-05-27. Полная история (v1.0, v1.0.1, v1.1, v1.1.1, v1.1.2 lock-in details, F1/F2 reconciliation history, Phase 5 migration plans, MVR migration plan, EN UPPERCASE migration log) — CHANGELOG.md.

ISO/IEC 29148 — матрица трассировки

> ***Назначение:*** *проверяемое соответствие заявления соответствия к ISO/IEC/IEEE 29148:2018 (standard/14 §14.4.2). Нормативные определения полей — в standard/06, standard/08, standard/09, 02-schemas.md.*

RENAR упрощает набор обязательных атрибутов 29148 (18 → 7–8 на артефакт) и добавляет TC как полноценный артефакт, ADAPT и SPEC-ось. Таблица ниже — **полная** трассировка для оценщика соответствия и для заполнения `external-claims[]` в манифесте.

1. Классы требований (29148 §5)

ISO/IEC 29148 класс	RENAR артефакт	Нормативный источник
Stakeholder requirement	`BR`	§6.5
System requirement	`SR` (level: system / subsystem / module)	§6.6
Software requirement (implementation unit)	`TR`	§6.7
Interface / design specification	`SPEC-*` (9 типов)	§8
Verification item	`TC`	§9
Requirements validation (client interpretation)	`ADAPT`	§7

2. Атрибуты требования (29148 Table B.1 → RENAR)

#	ISO/IEC 29148 атрибут	RENAR поле / механизм	Обязательность	Примечание
1	Unique ID	`id` (immutable)	обязательно	V1; см. §3.3.1
2	Requirement statement	body (Потребность / Поведение / ...)	обязательно	EARS-шаблон для SR: §6.6.3
3	Rationale	body «Контекст» + `source.adapt-section`	обязательно (BR/SR)	Прослеживаемость к ADAPT
4	Source	`source.adapt`, `source.tz-section`, `source.document-ref`	обязательно	V5 pinning через `document-ref`

5	Fit criterion	body «Критерии успеха» (BR) / Pass-критерии TC	обязательно	Измеримость через 25022/25023: §14.4.4
6	Priority	`priority` (MoSCoW)	обязательно	WSJF — informative в SAFe mapping
7	Owner	`owner` (BR/SR) / `business-context.stakeholder`	обязательно	§6.5.2
8	Status	`status` (enum жизненного цикла)	обязательно	Машины состояний: §10
9	Verification method	`verified-by[]` → `TC` + `tc-type`	обязательно для verify	TC как полноценный артефакт — расширение 29148
10	Parent / child	`parent.id` , auto `children[]`	обязательно (SR/TR)	Иерархия BR→SR→TR
11	Traceability (derived)	`verified-by` , `constrained-by[]` , `implements-spec[]` , KG edges	derived	reference/05 §4
12	Version	нативная для носителя версия + `requirement-version` в TC	обязательно (V5)	§3.3.5
13	Author	V6 author + `ai-provenance`	обязательно (RENAR-4+ AI)	§4.10.1
14	Date created / modified	timestamps записи изменений носителя	derived (V6)	Журнал аудита: §10.13
15	Risk	`compliance[]` , AIR register link	optional / domain	03-ai-risk-register.md
16	Assumption	ADAPT backward findings `type: assumption`	через ADAPT	§7.4.4
17	Dependency	`depends-on[]` (SPEC), `constrained-by[]` (SR)	обязательно где применимо	DAG invariant: 02 §9
18	Approval authority	QG-0 / QG-2 + ADAPT dual signature	обязательно	Замена formal walkthrough: §14.4.2

Не принято из 29148: review meetings и inspection-only verification без доказательной базы TC — см. §14.7.

3. Verification methods (29148 §6.4)

29148 method	RENAR реализация
Test	`TC` с `tc-type: system | acceptance | contract | …`

Demonstration	`tc-type: acceptance` + client sign-off (QG-4 optional)
Inspection	`[test-spec-change]` workflow + состязательный обзор (§9.13)
Analysis	SR с `quality-characteristic` + eval-TC (`tc-type: eval`) для SPEC-AI

4. Процессы жизненного цикла (29148 §6)

29148 process	RENAR глава	Gate
Requirements elicitation	ADAPT backward + T3	QG-0 (ADAPT approve)
Requirements analysis	BR/SR decomposition	QG-0 (BR/SR approve)
Requirements specification	SPEC axis	QG-3 Architecture (optional/required)
Requirements verification	TC + QG-2	QG-2 Verification
Requirements validation	ADAPT client signature + QG-4	QG-4 Acceptance (optional)
Requirements management	жизненный цикл §10 + носитель V1–V6	Continuous

5. Использование при оценке соответствия

1. Для каждого заявления `ISO/IEC/IEEE 29148:2018` в манифесте — пройти строки §2–§5.
2. Выборочно (≥10% артефактов или все BR/SR уровня system) проверить наличие обязательных полей и трассировку `source.adapt` .
3. Несоответствие любой **обязательной** строки §14 — частичное заявление недопустимо (§14.6.2).

Reference RENAR 1.0-draft — renar.tech

Самооценка соответствия

> ***Назначение:*** *практический kit для Tech Lead / Архитектор перед выпуском RENAR-CONFORMANCE.yaml. Нормативная база — standard/13. Этот документ* ***informative****; при расхождении побеждает standard/13.*

1. Взаимное соответствие MVR ↔ обязательные пункты §13.3

MVR (§0.5)	Положение §13.3	Поле `mandatory-clauses-confirmed`
MVR-1 инверсия источника истины	§13.3.1	`sot-inversion: true`
MVR-2 V1–V6	§13.3.2	`substrate-v1-v6: { v1..v6: true }`
MVR-3 ADAPT per T3	§13.3.3	`adapt-per-tz: true`
MVR-4 9 типов SPEC	§13.3.4	`spec-types-closed-list: true`
MVR-5 TC pos/neg	§13.3.5	`tc-pos-neg-pairing: true`
MVR-6 QG — закрытый список	§13.3.6	`quality-gates-closed-list: true`
MVR-7 манифест соответствия	§13.4 (артефакт)	манифест существует + подписан
— (политика закрытых списков)	§13.3.7	`closed-lists-backward-findings: true`

Все семь MVR + §13.3.7 обязательны для **любого** уровня RENAR-1..5.

2. Чек-лист самооценки (обязательные положения)

Отметьте после проверки доказательной базы в носителе:

§13.3.1 Инверсия источника истины

- ☐ Иерархия BR/SR/SPEC/TC — авторитетный источник о поведении
- ☐ Нет SR, восстановленных из кода без обоснования исправления дефекта
- ☐ Drift-hooks / политика обзора блокируют молчаливую адаптацию SR←код

§13.3.2 Возможности V1–V6 (носитель)

- ☐ V1 immutable history — включён
- ☐ V2 atomic change unit — включён
- ☐ V3 diff & review — включён
- ☐ V4 branching / change-set — включён
- ☐ V5 сквозная фиксация версии между носителями — включена (`verifies[].requirement-version`)

- [ ] V6 author + timestamp — включён

§13.3.3 ADAPT

- [ ] Каждое активное T3 имеет approved ADAPT
- [ ] Каждый delta-T3 имеет delta-ADAPT
- [ ] Двойная подпись (Архитектор + Клиент) зафиксирована

§13.3.4 SPEC types

- [ ] Все SPEC ∈ {ARCH, API, DATA, INT, PROC, UI, AI, SEC, OPS}
- [ ] Нет локальных `SPEC-CUSTOM-*`

§13.3.5 TC pos/neg

- [ ] Каждое верифицируемое утверждение имеет pos + neg TC (или исключение негативного инварианта)
- [ ] QG-2 блокирует `verified` при нарушении

§13.3.6 Quality Gates

- [ ] QG-0, QG-1, QG-2 реализованы как `required`
- [ ] QG-3, QG-4 объявлены `required` | `declared` | `absent` в манифесте
- [ ] Нет локальных пользовательских гейтов

§13.3.7 Закрытые списки

- [ ] Backward finding types — только закрытый список §7.4.4
- [ ] Типы декомпозиции SPEC — только закрытый список §8

Правило: хотя бы один не отмечен → манифест **не выпускать** (§13.5.1).

---

3. Чек-лист уровня (выберите целевой RENAR-N)

Минимум для заявления уровня — standard/11 §11.4–12.8. Краткая сводка:

Уровень	Ключевые доп. критерии
RENAR-1	Только обязательные положения; frontmatter минимален
RENAR-2	Канонический frontmatter + обеспечивается соблюдение статусов жизненного цикла
RENAR-3	Полная ось SPEC + hooks на все QG-0..QG-2
RENAR-4	ai-provenance обязателен; состязательный обзор для SPEC-SEC/AI
RENAR-5	Согласие нескольких моделей; непрерывная оценка; согласование KG

- [ ] Выбранный `level` в манифесте **не выше** фактически пройденного чек-листа

- ☐ `declared-stricter` (если есть) документирован отдельно

4. Шаблон манифеста (минимальный)

Сохранить как `RENAR-CONFORMANCE.yaml` в корне носителя требований:

```yaml
manifest-version: 1
manifest-id: "CFM-YYYY-NNN"
renar-version: "1.0-draft"
senar-version: "1.0"
level: RENAR-2
assessment-date: "2026-05-22"
assessment-type: self
next-assessment-due: "2026-08-22"

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: declared
  qg-4: absent

external-claims:
  - standard: "ISO/IEC/IEEE 29148:2018"
    scope: "requirements classes, attributes, lifecycle, verification"
    evidence: "reference/07-iso29148-trace-matrix.md"

substrate:
  type: git
  capabilities-verified: "2026-05-22"
  project-req-ref: "<нативный для носителя pointer>"

signed-by:
  name: "<Architect / Tech Lead>"
  role: approver
  signed-at: "2026-05-22T12:00:00Z"
```

Полный список полей — §13.4.2.

5. Периодичность

- Самооценка: **квартально** (по умолчанию)
- После delta-ТЗ с воздействием на обязательные положения — **внепланово**
- Триггеры потери соответствия — §13.8

Reference RENAR 1.0-draft — renar.tech

Педагогическая плотность (WC-02)

> ***Информативно.*** *Оценка плотности нормативного текста для калибровки читательского маршрута. Не изменяет обязательные положения.*

Метод (2026-05-22): `density score = (RFC-2119 RU markers + closed-list refs + state machine tables) / 1000 lines`, ручная калибровка + подсчёт строк. Шкала: **L** (low ≤3), **M** (3–6), **H** (6–9), **VH** (≥9).

По главам (оценки плотности)

Глава	Lines	Score	Tier	Signpost
00 Introduction	253	4.2	M	standard/README → начните с core
01 Scope	267	5.8	M	closed lists §1.7 — см. glossary
02 Methodology	241	5.0	M	Источник истины — core Rule 2 сначала
03 носитель V1–V6	246	5.2	M	→ guide/03 или guide/04
04 Terms	416	7.2	H	→ reference/01 для примеров
05 Roles	294	4.5	M	→ guide/01 walkthrough
06 Hierarchy	564	8.4	H	→ guide/00 quickstart перед frontmatter
07 ADAPT	355	6.8	H	→ core ADAPT section
08 Specifications	426	7.9	H	→ guide/09 E3 SPEC examples
09 Test cases	550	9.1	VH	→ reference/02 §8 + §9.1.1 decision tree
10 Жизненный цикл QG	628	9.5	VH	→ §10.1.1 QG tree + guide/00
11 Maturity	361	4.8	M	→ guide/02 transition
12 Metrics	361	3.9	M	provisional targets §12.4
13 Соответствие	394	8.0	H	→ reference/08 kit first
14 Normative refs	~220	2.8	M	→ reference/11 расширенный каталог; §14.5 «пять ключевых»

Топ-5 «горячих точек» (signposted)

1. **§10 Жизненный цикл** — см. quickstart QG flow
2. **§9 TC** — см. schemas §8 + состязательный §guide/07
3. **§6 Hierarchy** — см. quickstart + walkthrough
4. **§13 Соответствие** — см. reference/08 kit

5. **§8 SPEC** — см. guide/09 E3

Профиль реализации для агента

> ***Назначение:*** *informative contract для агента или нативного для носителя runtime, который* ***имплементирует*** *RENAR без догадок. Нормативный текст —* `standard/` *; этот документ — operational mapping без привязки к конкретному vendor tooling.*
> ***Машиночитаемо:*** `normative-index.yaml`

1. Как читать таблицу

Колонка	Смысл
`clause_id`	Стабильный ID из `normative-index.yaml`
`Действие агента (абстрактно)`	Что должен уметь runtime **без** vendor CLI
`Входы / выходы`	Артефакты носителя
`Gate`	Контрольная точка качества или проверка под управлением runner

2. MVR ↔ действия агента

clause_id	Действие агента (абстрактно)	Входы	Выходы	Gate
MVR-1	Блокировать reverse-engineering SR/SPEC из кода без bug-fix justification; источник истины = иерархия требований	code diff, SR/SPEC	audit record / blocked promotion	—
MVR-2	Проверить, что носитель декларирует V1–V6 в манифест; прогнать capability checks	`RENAR-CONFORMANCE.yaml`	pass/fail report	—
MVR-3	Реактивный стадийно-независимый ADAPT: создавать ADAPT (0..N на T3) только при findings состязательного обзора; нет findings → source.tz-section + adversarial-review-ref; delta-T3 → тот же реактивный паттерн; дезавуирование через superseded	TZ, состязательный вердикт, ADAPT draft	ADAPT approved / вердикт-свидетельство	QG-ADAPT-approve
MVR-4	Отклонять SPEC с type ∉ закрытого списка	SPEC frontmatter	validation error	QG-spec-approved
MVR-5	Обеспечить pos/neg пару TC на каждое нормативное утверждение	SR/SPEC, TC set	paired TC	QG-2

MVR-6	Отклонять custom gate types; требовать QG-0..2	project config	манифест	—
MVR-7	Требовать подписанный `RENAR-CONFORMANCE.yaml` с `senar-version`	манифест	claim соответствия	—

3. Взаимное соответствие MVR ↔ обязательные положения §13.3

Полное взаимное соответствие MVR ↔ §13.3 — `08-conformance-self-assessment.md §1` . Агент **не должен** считать проект соответствующим, если любое поле `mandatory-clauses-confirmed` = false.

4. Контракт gate runner (абстрактно)

Gate	Pre (проверки агента)	Post (записи агента)
QG-0	Schema valid; parent links; ADAPT exists for TZ-backed SR	`approved` transition + audit-trail
QG-1	TR links `implements-spec[]` ; носитель реализации pinned (V5)	TR `in_progress → done` evidence
QG-2	All `verified-by` TC `passing` ; pos/neg; `last-run.requirement-version` match	artifact → `verified`

Decision trees: standard/09 §9.1.1, standard/10 §10.1.1.

5. Правило нейтральности к носителю для implementer-ов

Runtime **должен** map-ить abstract actions на нативные для носителя primitives через `RENAR-CONFORMANCE.yaml#v1-v6-mapping` (§3.7). Vendor-specific команды **не** могут быть единственным способом выполнить обязательное положение.

6. Статус покрытия (v1.0-draft)

Область	Index entries	Profile rows	Примечание
MVR	7/7	7/7	complete
§13.3 обязательные	7/7	via bijection	complete
QG-0..2	3/3	3/3	QG-3/4 deferred optional
Обязательные положения по главам	partial	—	expand in later pass

Сопоставление с внешними стандартами

> ***Informative.*** *Детализация standard/14 §14.5. Не изменяет обязательные положения.*

1. SAFe 6.0

Framework масштабированного Agile. RENAR заимствует маппинг иерархии (§4.13.1): Portfolio Epic → BR, Feature → SR, Story → TR. Встроенное качество (TC до approved), WSJF для поля `priority`.

2. Spec-Driven Development

Индустриальный термин 2024–2025: при AI-ускорении критична корректность спецификации. RENAR формализует инверсию источника истины (§2.3.1) — нормативная структура, не привязанная к одному vendor-tool.

3. EARS (Mavin et al., 2009)

Шаблоны контролируемого естественного языка для SR (§6.6.3) и Pass/Fail в TC.

4. BDD / Gherkin / Specification by Example

Prior art для полноценного TC (§9): Given/When/Then, pos/neg как блокирующий gate, version pin V5.

5. NIST AI RMF 1.0

Govern / Map / Measure / Manage — функциональное сопоставление с ролями RENAR (§5), метриками (§12), жизненный цикл deprecate.

6. IEEE 830-1998 (deprecated)

Отозван в пользу ISO/IEC/IEEE 29148. Нормативный правопреемник — §14.4.2.

7. BABOK v3

Gap: elicitation вне scope (T3 уже зафиксировано); Solution Evaluation — частично QG-4 и §12.5.

8. PMBOK 7

«Принципы вместо процессов» — RENAR нормирует **что**, не **как** (§2.5).

9. ISTQB Foundation

Словарь тестирования; `tc-type` совместим с уровнями component/integration/system/acceptance.

10. CMMI v2.0

Prior art для уровней RENAR-1..5 (§11); process-heavy артефакты CMMI не базовый уровень.

11. ISO/IEC 42001:2023 (AIMS)

Организационный governance; RENAR даёт доказательную базу для requirements-среза (ai-provenance, манифест).

12. ISO/IEC 25059:2023

Расширение SQuaRE для AI; vocabulary для `quality-characteristic` AI-компонент.

13. EU AI Act (Reg. 2024/1689)

Поле `ai-act.risk-class` в BR; юридическое соответствие — вне RENAR.

14. SysML / MBSE

Prior art «требования как граф» — reference/05; RENAR выводит граф из текстовых артефактов.

Reference RENAR 1.0-draft — renar.tech