RENAR

Requirements Engineering & Normative Adaptive Regulation — нормативное ядро

RENAR · Стандарт · Версия 1.0 | 17.07.2026

Авторы: Вадим Соглаев, Андрей Юмашев

RENAR Core

Версия: 1.0 · **Дата:** 2026-07-14 · **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 (интерпретация ТЗ), ACTZ (протокол уточнения ТЗ), 11 типов SPEC (архитектура, API, данные, интеграция, процесс, UI, AI, безопасность, эксплуатация, тестовые стенды, поставляемая документация), TC (контрольные примеры) и AT (приёмочные тесты от контракта).
- **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-ТЗ 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 путь одного требования — от договора до приёмки:

```
flowchart TD
    K[Клиент] --> TZ[ТЗ — договор, неизменяемое]
    TZ --> AR{Состязательный обзор}
    AR -->|«есть находки»| AD["ADAPT — интерпретация:<br/>как ТЗ прочитано инженерами;<br/>подпись архитектора"]
    AR -->|«находок нет»| ART["BR / SR / SPEC / TR / TC<br/>(RENAR-описание)"]
    AD -->|вопрос клиенту| ACTZ["ACTZ — «Протокол уточнения ТЗ»:<br/>решения на языке обязательств;<br/>подписи клиента и исполнителя"]
    ACTZ --> AD
    AD --> ART
    ART --> IMPL[Реализация код]
    TZ --> EFF[Итоговое ТЗ = ТЗ + подписанные протоколы]
    ACTZ --> EFF
    EFF --> AT["AT — приёмочные тесты<br/>от контракта"]
    IMPL --> AT
```

Ключевые свойства:

- **ТЗ — договорной неизменяемый артефакт.** После подписания клиентом оно не редактируется. Изменения объёма работ формализуются через delta-ТЗ; уточнения — через протоколы (см. ниже).
- **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».
- **Состязательный обзор.** Отдельный AI-агент с другой моделью специально ищет, что первичный агент пропустил: недостающие вопросы к клиенту, ослабленные критерии тестов, скрытые допущения. Это компенсирующий механизм против самосогласованных, но семантически неверных AI-выводов.
- **Парность тестов.** Каждое проверяемое утверждение требования имеет минимум один положительный и один отрицательный тест-кейс. AI-агенты охотно покрывают благополучный сценарий и обходят отрицательные — RENAR делает парность нормативной.

- **Тест пишет не тот, кто пишет код, и тест обязан хоть раз побывать красным.** Тест, сочинённый вместе с реализацией, проверяет то, что написано, а не то, что требовалось. А тест, никогда не падавший, не доказал ничего: возможно, он просто пуст.

Два документа: интерпретация и утверждение

Самая частая ошибка в договорной разработке — смешать в одном документе то, **как инженеры поняли ТЗ**, и то, **о чём договорились с клиентом**. Смешение выглядит экономным, но ломает обе функции сразу: клиент подписывает инженерный текст, который не читал и не в состоянии оценить, а инженер боится записать честную трактовку, потому что её понесут на подпись.

RENAR разводит эти два документа, и граница проводится **по аудитории**, а не по содержанию:

> ***Всё, что показано клиенту и утверждено, — обязательство. Всё, что не показано, — интерпретация.***

	ADAPT — интерпретация	**ACTZ — утверждение**
Что внутри	Как ТЗ прочитано: перевод на инженерный язык, сопоставление терминов, достроенные сценарии, найденные пробелы и противоречия	Вопросы и решения, вынесенные клиенту. На языке обязательств: «кнопка называется так», «срок — такой»
Кто читает	Инженер, AI-агент, рецензент, верификатор	Клиент и стороны договора
Кто подписывает	Только архитектор со стороны исполнителя	**Обе стороны**: клиент и исполнитель
Вес	Внутренний рабочий документ	**Договорной**

ACTZ — тот самый «**Протокол уточнения ТЗ № N**», который в договорной практике и так существует. RENAR лишь делает его обязательным местом, где живут решения клиента, и связывает: каждая находка в интерпретации, потребовавшая слова клиента, обязана указывать пункт **подписанного** протокола, в котором это слово записано. Ни одна трактовка не может опираться на решение, которого клиент не принимал; и ни одно подписанное решение не может остаться неотражённым в требованиях.

Выигрыш прост и проверяем машиной: спор «это уточнение или уже изменение объёма?» исчезает. Вопрос теперь бинарный — **выносилось ли решение клиенту и подписано ли оно**.

Приёмка — от контракта, а не от интерпретации

Отсюда следует **итоговое ТЗ**:

> ***Итоговое ТЗ = начальное ТЗ (с приложениями) + все подписанные протоколы уточнения.***

Именно оно — эталон сдачи-приёмки. Внутренняя интерпретация в эталон **не входит**: клиент отвечает только за то, что подписывал и понимал. Приёмка становится юридически чистой.

Но у этого есть и инженерное следствие, ради которого всё и затевалось. Обычные тесты выведены из требований, а требования — из интерпретации. Значит, если интерпретация неверна, все тесты могут быть зелёными: система безупречно соответствует **неверному** пониманию ТЗ — и

проваливает приёмку у заказчика. Проверять систему тем же пониманием, которое её построило, — значит гарантированно не увидеть ошибку понимания.

Поэтому RENAR вводит второй, независимый слой проверки — **приёмочные тесты (AT)**:

- их пишет **изолированный** агент, которому на вход дают **только итоговое ТЗ**: ни интерпретации, ни требований, ни спецификаций, ни кода;
- модель этого агента отличается от модели основного — иначе он воспроизведёт ту же трактовку, и приёмка выродится в самоподтверждение;
- перед каждыми испытаниями тесты пересобираются от **действующей** редакции итогового ТЗ: длинный заказ иначе сдаётся против контракта годичной давности;
- продукт не предъявляется к сдаче, пока все приёмочные тесты не зелёные.

Расхождение двух слоёв — не шум, а диагноз. **Приёмочный тест красный, а обычные зелёные — это ошибка интерпретации**: система сделана правильно, но не то. Такой дефект не ловится никаким количеством обычных тестов, и в этом весь смысл второго слоя.

Полный жизненный цикл нормирован через **контрольные точки качества**: QG-0 (утверждение), QG-1 (реализация), QG-2 (верификация) обязательны; QG-3 (архитектура) и QG-4 (бизнес-результат) опциональны. Отдельно от них стоит релизный гейт приёмки: соответствие контракту доказывается до предъявления продукта.

Штатный исполнитель — AI-агент

RENAR-артефакты **штатно** создаются и поддерживаются AI-агентом по заданию инженера. Человек выполняет роль проверяющего и утверждающего: 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** там, где возникает обязательство: подпись протокола уточнения (клиент и исполнитель), подпись интерпретации (архитектор), утверждение QG-0, выборочная проверка тестов, приёмка результата. AI-агент исполняет; человек отвечает за результат. Клиент при этом не общается с агентом напрямую: вопросы агрегирует и переформулирует архитектор.

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 исполнитель + клиент с подписанным ТЗ и критериями приёмки.

- **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 ТЗ 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: ТЗ → 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*** · *← 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, ACTZ, одиннадцати типов SPEC, TC, AT.
- **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** — двусторонняя подпись ACTZ (§7.13), подпись архитектора на ADAPT (§7.5), изоляция авторства тестов и красная история (§9.18), выборочная проверка инженера (§9.14), состязательный обзор (§7.10.2), 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 / ACTZ / 11 SPEC types / TC / AT (§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 подписью архитектора; ACTZ с двусторонней подписью и итоговое Т3 как эталон приёмки; 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 требований. На одно T3 приходится **ноль или более** ADAPT; каждый привязан к своему триггеру (импорт T3 либо конкретная стадия декомпозиции) через `trigger-stage`, множественность — штатный случай. При наличии gap ADAPT обязателен в статусе `approved` с подписью архитектора; решения клиента фиксируются подписанными ACTZ (§7.13), а эталоном сдачи-приёмки служит итоговое T3 = начальное T3 + все подписанные ACTZ (§7.14). При отсутствии gap (состязательный рецензент вынес вердикт «no findings») ADAPT не создаётся; BR/SR/SPEC ссылаются на T3 напрямую через обязательные `source.tz-section` + `source.adversarial-review-ref` → AR (§7.4.6). Исход обзора в обоих случаях выпускается как AR. Delta-T3 следует тому же правилу.	§7, §13.3.3
MVR-4	**Закрытый список 11 типов SPEC**: тип SPEC обязан принадлежать закрытому списку `SPEC-ARCH`, `SPEC-API`, `SPEC-DATA`, `SPEC-INT`, `SPEC-PROC`, `SPEC-UI`, `SPEC-AI`, `SPEC-SEC`, `SPEC-OPS`, `SPEC-TEST`, `SPEC-DOC`.	§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

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 + подпись архитектора; ACTZ (протокол уточнения ТЗ) с двусторонней подписью	07
3	Закрытый список 11 типов спецификаций (SPEC-ARCH / API / DATA / INT / PROC / UI / AI / SEC / OPS / TEST / DOC)	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 (§7.10.2).
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 приёмку (§10.4.2) и подписать ACTZ (§5.5).
3. **Реактивная двусторонняя client-side validation** — состязательный обзор ТЗ обязателен; если обнаружены findings или требуется уточнение, вопросы выносятся клиенту протоколом ACTZ (§7.13), а его решения фиксируются в 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; критерии приёмки (QG-4 §10.4.2) не формализуемы — критерий «успеха» научного исследования не подписывается заранее; двусторонняя подпись ACTZ 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 двусторонней подписи ACTZ. При отсутствии независимого представителя клиента двусторонняя подпись ACTZ (§5.5) структурно невозможна — `client-signature.signed-by == vendor-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 ACTZ.
Проект, в котором scope пересматривается без формальной фиксации delta-ТЗ	Нарушение §7.6 delta-ADAPT workflow; нарушение §13.3 обязательное положение §13.3.3 «реактивный ADAPT: состязательный обзор на каждое ТЗ, ADAPT — по вердикту».
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 (5 признаков + 5 типичных представителей)	§1.4	§1.7.1
4	Negative scope: контексты неприменимости (4 контекста + 4 негативных сценария)	§1.5	§1.7.1
5	Роли RENAR (specializations над SENAR §4)	§5	§1.2 (10)
5bis	Типы оси требований (3 типа: BR / SR / TR)	§6.2	§13.3
5ter	Уровни декомпозиции (3: система / подсистема / модуль)	§6.3	§6.4, §4.9
6	Категории backward findings ADAPT (7 категорий)	§7.4.4	§13.3.7
7	Типы спецификаций SPEC-* (11 типов: ARCH/API/DATA/INT/PROC/UI/AI/SEC/OPS/TEST/DOC)	§8.3	§4.4.1, §13.3.4
8	Нормативные принципы TC (закрытый список, P1–P9)	§9.2	§13.3
9	Типы TC (`tc-type` : 6 значений; `business` вместо прежнего `acceptance`)	§9.5	§4.5.2
10	Состояния жизненного цикла по типам артефактов (BR / SR / TR / SPEC / ADAPT / TC)	§10.5–§10.9	§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 (11 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`	Концептуальный обзор стандарта для человека-читателя (не нормативный).
`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 §3.5) 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 T3 + подпись архитектора; ACTZ с двусторонней подписью (§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: T3 → 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:

```
flowchart TD
    U1["Утверждение 1 — инверсия источника истины<br/>требования > код"]
    U3["Утверждение 3 — версионирование носителя V1–V6<br/>provenance, pin,
history, atomic change"]
    U2["Утверждение 2 — waterfall-форма, не классика<br/>дельта-Т3, парные ТС,
ADAPT, reconciliation"]
    KOR["Контракт-ориентированная разработка<br/>жизнеспособна с AI-ускорением"]
    U1 -->|требует| U3
```

```
    U3 -->|позволяет| U2
    U2 -->|возвращает| KOR
```

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 цепи 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 инверсии 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*** · *← 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-0, QG-ADAPT-approve
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 (подпись архитектора + переход `answered → approved` за одну транзакцию); подписание ACTZ — отдельное атомарное действие с двусторонней подписью.
- 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-0 — утверждение требований и спецификаций (§10.3.1); QG-ADAPT-approve — утверждение ADAPT (§7.9).
- Подпись архитектора на ADAPT (§7.5) и двусторонняя подпись ACTZ (§7.13).
- 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:

- Подпись архитектора на ADAPT и двусторонняя подпись ACTZ (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`»
«Двусторонняя подпись ACTZ — два независимых события автор+отметка времени (V6)»	«Двусторонняя 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).

```yaml
# Фрагмент RENAR-CONFORMANCE.yaml — substrate-специфичная часть
# (полная схема — §13.4.2)
substrate-capabilities:
  v1-immutable-history:  declared    # primitive носителя и способ проверки — в substrate-id
  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>"
```

Подтверждение обязательных положений (§2.3 инверсия источника истины, §7 ADAPT, §8 закрытый список 11 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 ADAPT подписью архитектора — V3 + V6; двусторонняя подпись ACTZ — 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*** · *← 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`) — **реактивный** мостовой артефакт (§7.4.1) между ТЗ и иерархией требований. Содержит forward (инженерная интерпретация) + backward (вопросы клиенту) разделы. Создаётся тогда и только тогда, когда состязательный обзор вынес вердикт «findings present» на любой стадии деривации; тогда ADAPT обязан быть в статусе `approved` с подписью архитектора (§7.5); клиент ADAPT не подписывает — его решения живут в ACTZ (§4.3.7). На одно ТЗ приходится **ноль или более** корневых ADAPT (кардинальность 0..N, MVR-3, §13.3.3); при вердикте «no findings» ADAPT не создаётся, и артефакты ссылаются на ТЗ напрямую (§4.10.4 AR). 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 ограничен спецификациями через `constrained-by[]` (§8.6.1) — это **связь**, а не parent-ребро; ребро `implements-spec[]` принадлежит TR (§8.6.2).

4.3.7 ACTZ — протокол уточнения ТЗ

ACTZ (`ACTZ-NNN` , Agreed Clarification of TZ) — артефакт **контрактного контура**: порция вопросов, предложений и решений, вынесенная клиенту и подписанная **обеими сторонами** (§7.13).

Договорное имя — «**Протокол уточнения ТЗ № N**». Несёт контрактный вес; после подписания неизменяем. Кардинальность: 0..N на ТЗ, 1..N на ADAPT. Жизненный цикл: `draft → sent → signed → superseded`.

Граница с ADAPT — **по аудитории**: показано клиенту и утверждено — обязательство (ACTZ); не показано — интерпретация (ADAPT).

4.3.8 Итоговое ТЗ (effective TZ)

Итоговое ТЗ — эталон сдачи-приёмки (§7.14): начальное ТЗ (включая приложения) **плюс все подписанные ACTZ**. При расхождении приоритет у более позднего подписанного документа. Из итогового ТЗ выводятся приёмочные тесты AT (§4.5.4). ADAPT в эталон приёмки не входит.

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 11 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) — **как система живёт**
`SPEC-TEST`	Тестовые стенды и данные (топология, эмуляторы, датасеты по обе стороны интеграций, правила сверки) — **как доказывается соответствие** (§8.3.2)

`SPEC-DOC`	Поставляемая документация (состав, структура, проверяемые требования) — **что входит в поставку**

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, §9.3). Жизненный цикл: §4.6.5.

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

Закрытый список (§9.5):

`tc-type`	Назначение
`business`	E2E-тесты бизнес-цели для BR (§9.5); runner-family: E2E + AI-валидатор. Прежнее имя `acceptance` изъято: приёмок две, и одно имя на два предмета путало
`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).

4.5.4 AT — Acceptance Test (приёмочный тест контрактного контура)

AT (`AT-NN`) — приёмочный тест, выводимый **изолированным агентом исключительно из итогового ТЗ** (§4.3.8), без доступа к ADAPT / BR / SR / SPEC / TC / коду (§9.19). Проверяет соответствие **контракту**, а не интерпретации. Перегенерируется перед каждыми испытаниями от действующей редакции итогового ТЗ. Расхождение «AT красный при зелёных TC» диагностирует **ошибку интерпретации** — класс дефектов, который TC не ловит в принципе.

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 → asked → answered → approved → frozen` , и терминальное `superseded` при дезавуировании (§10.8.5). `frozen` — терминальный immutable; изменения только через delta-ADAPT, errata или дезавуирование.

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 (подпись архитектора) / 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 TC 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 TZ до прогона TC, фиксирующая происхождение каждого утверждения:

```
TZ → 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).

4.10.4 AR — запись состязательного обзора

AR (`AR-NNN` , Adversarial-Review Record) — неизменяемая после выпуска **запись-свидетельство**, фиксирующая факт и исход обязательного состязательного обзора TZ (§7.4.6). Обязательные поля: `tz-ref` , `trigger-stage` , `reviewer` (модель, отличная от модели основного агента),

`verdict` (`findings-present` либо `no-findings`), `produces-adapt[]` (при `findings-present`), `status`, подпись V6. Жизненный цикл: `draft → issued → superseded`.

AR — **не** артефакт требований и **не** узел графа: она не декомпозируется, не верифицируется тест-кейсами и не входит ни в один закрытый список стандарта (§1.7.5). Её назначение — дать вердикту состязательного обзора идентичность и сделать его аудируемым в обоих исходах: при `findings-present` AR указывает на порождённый ADAPT, при `no-findings` — сама служит свидетельством, на которое ссылается `source.adversarial-review-ref` (§4.12).

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` (§9.3)
`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-*` (11 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*** · *← 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: **двусторонняя подпись ACTZ** (клиент + исполнитель), без которой решение клиента не является обязательством. ADAPT при этом подписывает только Архитектор — это внутренняя интерпретация.

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 ADAPT (§7.5) и исполнительская сторона двусторонней подписи ACTZ (§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: ACTZ (§7.13) 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, подписи 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 двустороннюю подпись ACTZ §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)	Единоличная — внутренняя интерпретация	QG-0: Архитектор; QG-3 (опц.): подпись архитектора (§7.5)
5.3.1bis	**ACTZ**	Архитектор (исполнитель) + представитель клиента	Совместная — обе подписи обязательны	Подписание (§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
5.3.5bis	**AT**	Архитектор (исполнитель)	Разделённая: авторство изолировано (§9.19.2) — генерирует изолированный AI-агент на другой модели; привязку к стенду (`environment-ref`) и прогон ведёт сторона исполнителя	Релизный гейт приёмки AT (§10.4.3): автоматический runner
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

ACTZ — единственный артефакт RENAR с **обязательной** совместной ответственностью: решение не становится обязательством без подписи обеих сторон (§7.13) — нормативная защита от односторонней фиксации обязательств.

ADAPT — артефакт единоличной ответственности Архитектора (§7.5): это внутренняя интерпретация, и клиент не подтверждает то, чего не читает. Защита от односторонней интерпретации ТЗ обеспечивается не подписью клиента под ADAPT, а тем, что **каждая находка обязана быть закрыта подписанным ACTZ** (§7.4.5): интерпретация не может опираться на решение, которого клиент не принимал.

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), TEST (test-

environment lead), DOC (documentation 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 ACTZ подписывается односторонне (без представителя клиента) — нарушение §5.3.1 и §13.3; манифест несоответствующий. Обратно: требование клиентской подписи под ADAPT избыточно и базовым соответствием не является — это `declared-stricter` расширение (§1.7.2).

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 ACTZ (двусторонняя 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 ACTZ

5.5.1 Нормативное правило

Двусторонняя подпись стоит на **ACTZ** — протоколе уточнения ТЗ (§7.13), а не на ADAPT. ACTZ переходит в `signed` **только** при наличии обеих подписей:

1. **`client-signature`** — со стороны представителя клиента (заинтересованная сторона с полномочиями подписания от клиента).
2. **`vendor-signature`** — со стороны исполнителя.

ADAPT подписывает только Архитектор (§7.5): это внутренний артефакт интерпретации, клиент его не видит.

Граница проводится по аудитории: всё, что вынесено клиенту и утверждено, — обязательство (ACTZ, двусторонняя подпись); всё, что не вынесено, — интерпретация (ADAPT, подпись архитектора). Клиентская подпись под ADAPT была бы фикцией — клиент подписывал бы инженерный документ, который не читает и не может оценить.

Структура полей подписей фиксирована в §7.13.3; 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` (на ACTZ)	Решения, вынесенные на утверждение, приняты; формулировки обязательств («будет сделано так-то») соответствуют намерению клиента; решения финальные.
`vendor-signature` (на ACTZ)	Исполнитель принимает решения как обязательства и берётся их реализовать.
`architect-signature` (на ADAPT)	Прямая интерпретация технически реализуема; все обратные находки в состоянии `resolved` и несут `decided-in` на пункт подписанного ACTZ (§7.4.5); декомпозиция в BR / SR / SPEC безопасна для запуска.

Подписи **не** взаимозаменяемы. Архитектор не подтверждает клиентскую семантику; клиент не подтверждает техническую реализуемость и не берёт на себя ответственность за интерпретацию.

5.5.3 Негативные сценарии

- **Одна подпись отсутствует на ACTZ**: протокол не переходит в `signed`; решения, которые он несёт, обязательствами не являются; находки, ссылающиеся на него через `decided-in`, не могут быть `resolved`, и ADAPT не переходит в `approved` (§7.4.5).
- **Одно лицо в обеих подписях ACTZ**: реализация, в которой `client-signature.signed-by == vendor-signature.signed-by`, нарушает §5.5.1 (две стороны требуют двух независимых лиц). Сценарий внутреннего продукта без независимого представителя клиента — вне основной области применения (§1.5.4).
- **Подпись клиента на ADAPT**: избыточна и нормативно **не требуется**. Реализация, требующая её, — `declared-stricter` расширение (§1.7.2), а не соответствие базовому уровню.
- **Уполномоченное лицо вместо Архитектора**: допустимо (§5.4.2); подпись фиксируется с `role: authorized-role-holder`.
- **Уполномоченное лицо вместо Клиента**: **не допустимо** (§5.4.3); `client-signature` обязана быть от заинтересованной стороны на стороне клиента.
- **Подписанное решение без отражения в интерпретации**: ACTZ подписан, но ни один ADAPT его решения не несёт — обязательство существует вне требований; **fatal** (§7.4.5).

5.5.4 Связь с другими gates

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

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 ТЗ	R	A	—	—	—	C (состязател
Создание ADAPT (forward + backward)	R	A	C	—	—	C (состязател

Утверждение ADAPT (подпись архитектора)	—	**A (`architect-signature`)**	—	—	—	C (состязател
Подписание ACTZ (двусторонняя подпись)	—	**A (`vendor-signature`)**	—	—	—	—
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	—
Генерация AT из итогового ТЗ	**R (изолированный агент, другая модель)**	A	—	—	—	—
Привязка AT к стенду (`environment-ref`)	—	**A**	—	R	—	—
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`)	Подпись Архитектора; все находки закрыты подписанными ACTZ	Архитектор	AI-критик	Команда
QG-4 Acceptance Gate (опционально, BR)	BR (`verified → accepted`)	Заинтересованная сторона (Клиент + ВП)	ВП	Архитектор	Команда
Релизный гейт приёмки AT (обязателен, не QG — §10.4.3)	AT (все в `passing`); продукт	Автоматический runner	Архитектор	—	Команда, Клиент

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 двусторонней подписи ACTZ (§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 двустороннюю подпись для BR/SR (не только для ACTZ); требовать подпись клиента под 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 ACTZ подписанным 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 поля `architect-signature` ; `client-signature` и `vendor-signature` — на ACTZ (§7.13.3)
§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	Подпись Архитектора на ADAPT (§7.5); двусторонняя подпись (§5.5) — на ACTZ
§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*** · *← 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 11 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 ноль или более корневых ADAPT — по числу находок состязательного обзора, §7.4.1.4).

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: AR-NNN   # conditional: present если source.adapt отсутствует — AR с verdict «no findings» (§7.4.6)

# === 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 указывает на запись состязательного обзора AR (§7.4.6), хранящую этот вердикт 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` указывает на существующую AR в статусе `issued` с вердиктом `no-findings`.

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)

```
---
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
```

```
  tz-section: "§N.N"                  # mandatory всегда
  adversarial-review-ref: AR-NNN    # mandatory если adapt omitted (§7.4.6)

# === Характеристика качества (conditional) ===
# mandatory для нефункционального SR; значение — из перечня ISO/IEC 25010:2023
# (§14.4.3). Для функционального SR поле опускается.
quality-characteristic: functional-suitability | performance-efficiency |
compatibility |
                        interaction-capability | reliability | security |
maintainability |
                        flexibility | safety

# === 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.
```

```yaml
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 T3 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 T3), 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 T3 → RENAR порождает разрыв между языками. Цепочка прослеживаемости соответственно имеет **два валидных варианта**, выбираемых в зависимости от того, существует ли ADAPT для конкретного T3.

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

```
TC-NN  →  verifies SR-12 v1.4
              │
              ├─ parent:        BR-03 v2.0     (BR-03 — level: subsystem)
              │                     │
              │                     └─ implements: BR-01 (system), BR-05 (system)
              │                                          (типизированное
межуровневое ребро §6.8.2)
              ├─ source.adapt:  ADAPT-001 §Forward §3.2
              │     └─ source-tz: TZ-2026-001 §3.4
              ├─ 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: AR-002
              ├─ source.tz-section: TZ-2026-001 §3.4   (нет ADAPT для этого T3)
              │  source.adversarial-review-ref: AR-002
              ├─ 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; ссылка `adversarial-review-ref` ведёт на запись состязательного обзора AR (§7.4.6), которая фиксирует, кем и когда было задекларировано «no findings» (V6 author + timestamp), и доступна 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 цепочке 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 ТЗ
  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 Требование, порождённое реализацией (`implementation-originated`)

6.13.1 Нормативное правило

Инверсия источника истины (§2.3.3) запрещает выводить требование из наблюдаемого поведения кода и подгонять SR под уже написанное. Запрет **не ослабляется**: он входит в MVR-1.

Но у запрета есть цена, которую он не должен взимать. AI-агент, реализуя задачу, штатно добавляет **внутренние технические детали**, которых нет в требованиях: защитную проверку входа, обработку граничного случая, журналирование. Требовать на каждую такую деталь delta-ADAPT с подписью клиента — абсурдная церемония, и именно она толкает к молчаливой подгонке требований задним числом.

Поэтому вводится **узкий** легальный класс: `source: implementation-originated` .

6.13.2 Граница класса

Что	Куда
Наблюдаемое клиентом поведение (влияет на приёмку: экран, поле, сообщение, срок, формат, объём поставки)	**Только** через контрактный контур: ADAPT / ACTZ с подписями. Послаблений нет
Внутренняя техническая деталь без наблюдаемого клиентом поведения (защитная проверка, внутренняя валидация, журналирование, обработка граничного случая)	Допустим `source: implementation-originated`

Граница определяется **наблюдаемостью для клиента**, а не размером правки. Сомнение трактуется в пользу контрактного контура.

6.13.3 Обязательная обвязка

Требование класса `implementation-originated` **обязано** нести:

1. **Провенанс**: ссылку на единицу изменения реализации, которая его породила, и обоснование — почему функциональность признана нужной.
2. **Утверждение человека**: явное одобрение Супервизора (человека, не агента). Агент не может легализовать собственную добавку.
3. **ТС до слияния**: покрывающий тест обязан существовать и проходить до включения изменения.
4. **Мутационную проверку** (§9.18.2): у такого ТС **не может быть красной истории** — он пишется после кода и рождается зелёным. Его зубастость доказывается **убитым мутантом**; без убитого мутанта ТС доказательством не является.
5. **Счётчик в метриках дрейфа** (§12.3): рост доли `implementation-originated` — сигнал, что процесс требований протекает, и он обязан быть виден.

6.13.4 Что остаётся запрещённым

- Молчаливая правка существующего SR под наблюдаемое поведение кода — по-прежнему **нарушение** (§2.3.3, MVR-1). Класс `implementation-originated` легализует **добавление** внутренней детали с провенансом и подписью человека, а не **переписывание** нормы под факт.
- Класс `implementation-originated` для наблюдаемого клиентом поведения — **несоответствие**: обязательство перед клиентом не может возникнуть из кода.

6.14 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 ТЗ 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

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

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 выносятся клиенту и возвращаются его решениями.

Но документ, который читает клиент, и документ, которым живёт инженер, — **разные документы**. Решения, вынесенные клиенту и подписанные, — это обязательства, и они живут в отдельном артефакте: **ACTZ**, протоколе уточнения ТЗ (§7.13). ADAPT остаётся внутренней интерпретацией: его подписывает только архитектор, клиент его не видит. Граница проводится не по содержанию записи, а **по аудитории** — всё, что показано клиенту и утверждено, есть обязательство; всё, что не показано, есть интерпретация. Так исчезает неразрешимый спор «это толкование или уже изменение».

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»).

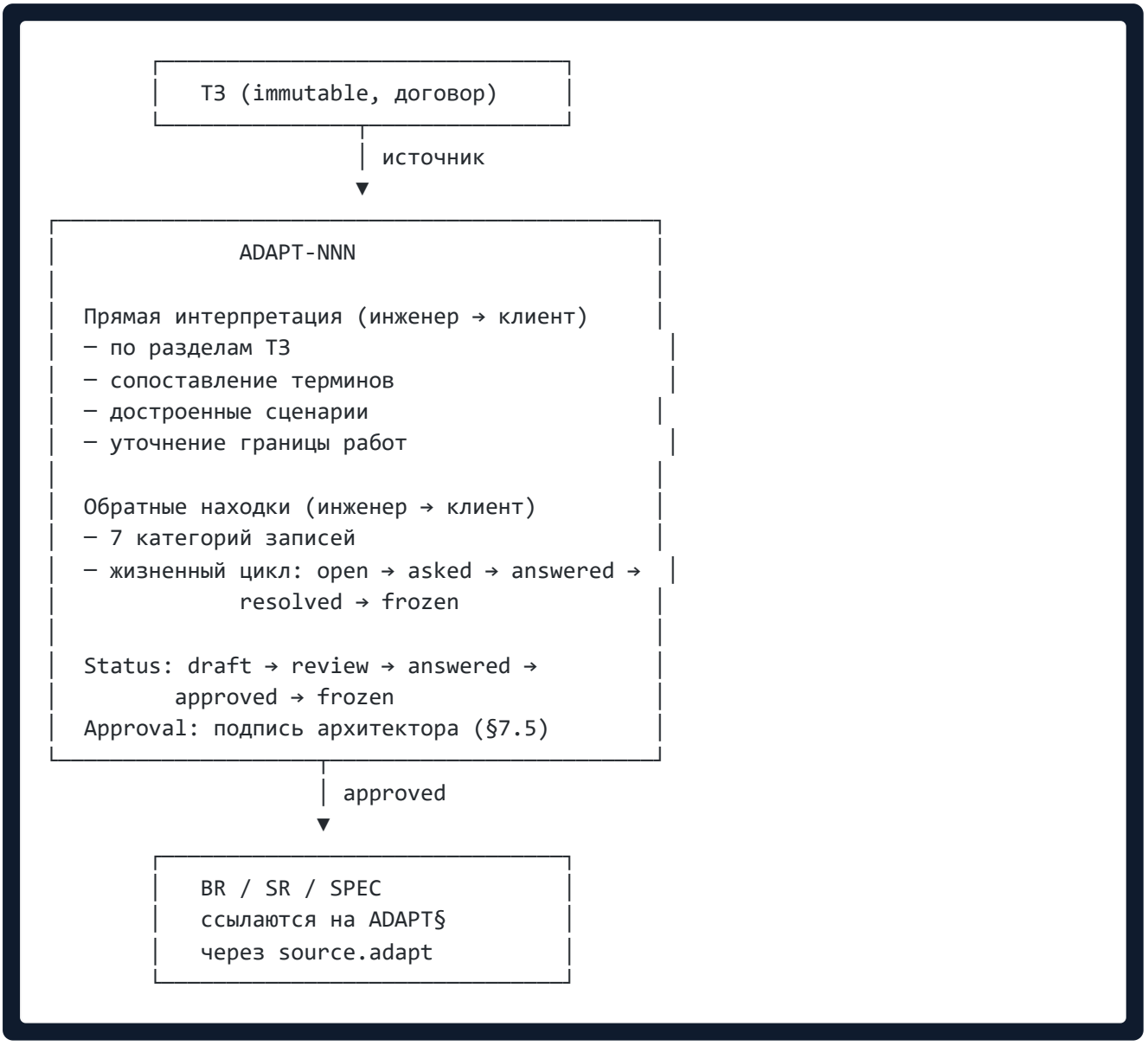
7.2 Проблема, которую нормирует ADAPT

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 подписи архитектора (§7.5), а решения клиента — в подписанных обеими сторонами ACTZ (§7.13).

7.3 Цикл двусторонней адаптации



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

Показанный вход «ТЗ → ADAPT» — типовой (импорт ТЗ), но **не единственный**. Триггер цикла стадийно-независим: если вопрос к клиенту с корнем в ТЗ всплывает позже — на стадии декомпозиции BR → SR → SPEC или при разработке ТС, — цикл запускается повторно и порождает **новый** ADAPT-NNN, привязанный к стадии, на которой находка обнаружена (§7.4.1.1). На одно ТЗ 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 ТЗ → RENAR-описание порождает разрыв между языком клиента и языком требований (§7.12). Если конвертация однозначна — ADAPT не создаётся, BR / SR / SPEC выводятся напрямую из ТЗ 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 ТЗ 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 ТЗ.
2. **Требуется сопоставление терминов (term mapping).** ТЗ использует термин, не имеющий однозначной инженерной интерпретации (требует сопоставления «клиент → инженер»).
3. **Требуется уточнение границы работ.** Граница работ из ТЗ 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 ТЗ напрямую через `source.tz-section` (§6.5.2, §6.6.2, §8.5).

7.4.1.2 Состязательный рецензент как обязательный гейт

Состязательный обзор ТЗ (§7.10.2) — **обязательный шаг** для каждого ТЗ 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 ТЗ. Вердикт «no findings» при импорте не блокирует появление ADAPT позже, если на стадии декомпозиции обнаружится находка с корнем в ТЗ (§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 ТЗ	ADAPT обязателен; жизненный цикл §7.4.5 запускается
«no findings, no clarifications»	Подтверждение состязательного рецензента (другая модель, §9.4) что ТЗ конвертируется в RENAR однозначно	ADAPT можно не создавать; вердикт остаётся зафиксированным свидетельством

В обоих исходах вердикт обязан быть зафиксирован в **записи состязательного обзора** (AR, §7.4.6) — неизменяемой после выпуска записи-свидетельстве с собственным идентификатором, автором и отметкой времени (V6). AR — единственный носитель вердикта: при исходе «findings present» она указывает на порождённый ADAPT, при исходе «no findings» она **сама** является тем свидетельством, на которое ссылается поле `source.adversarial-review-ref` (§6.5.2).

Hook (§10.11.1 `adapt-applicability validation`) при создании BR / SR / SPEC без `source.adapt` проверяет наличие AR со статусом `issued` и валидным вердиктом для

соответствующего ТЗ. Отсутствие такой записи — fatal: создание блокируется до прохождения состязательного обзора.

7.4.1.3 Запрет молчаливого пропуска

Создание BR / SR / SPEC из ТЗ **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 выпущен как AR (§7.4.6) с автором и отметкой времени (V6) и доступен по запросу аудитора (§13.5).

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

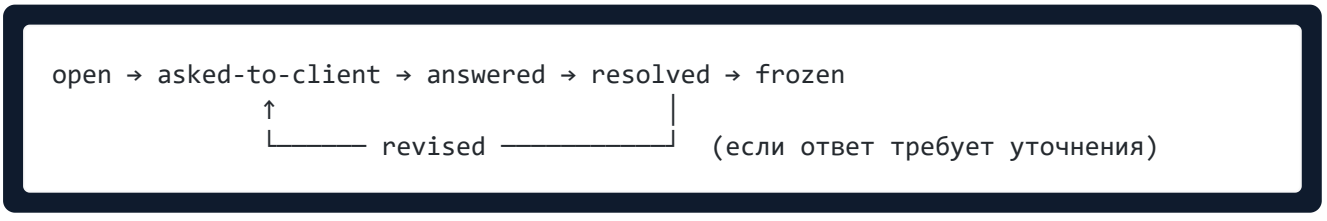
7.4.4 Обратные находки и их категории

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



Статус	Что значит
`open`	Инженер записал; не вынесено клиенту

`asked-to-client`	Вопрос вынесен клиенту в составе ACTZ (§7.13); ожидается подписание; зафиксирована дата
`answered`	Клиент принял решение; решение зафиксировано пунктом **подписанного** ACTZ
`resolved`	Инженер интегрировал решение в прямую интерпретацию; запись несёт `decided-in: ACTZ-NNN §M`
`revised`	Решение клиента расплывчатое; повторный вопрос — новым ACTZ. Переход обратно в `asked-to-client`
`frozen`	После approval ADAPT — изменения невозможны

Связь с ACTZ обязательна. Каждая запись, требующая решения клиента, обязана нести поле `decided-in: ACTZ-NNN §M` — ссылку на пункт **подписанного** ACTZ, в котором решение зафиксировано (§7.13). Ссылка на неподписанный ACTZ, а равно висячая ссылка на несуществующий ACTZ — **fatal** (§10.11.1).

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 — то есть каждая обязана иметь подписанное решение клиента.

Обратное направление тоже обязано быть выражено: решение, принятое клиентом по его собственной инициативе (ACTZ без родительской находки, §7.13.2), обязано быть **отражено в интерпретации** — ADAPT реагирует на ACTZ. Подписанное решение, не отражённое ни в одном ADAPT, — **fatal**: это обязательство, которого нет в требованиях.

7.4.6 AR — запись состязательного обзора

AR (Adversarial-Review Record, запись состязательного обзора) — неизменяемая после выпуска запись-свидетельство, фиксирующая факт и исход обязательного состязательного обзора (§7.4.1.2).

AR принадлежит классу **свидетельств**, а не узлов графа требований: у неё нет родителей и потомков, она не декомпозируется, не верифицируется тест-кейсами и не является ни требованием, ни спецификацией. Поэтому AR **не расширяет** ни один закрытый список стандарта (§1.7.5): она доопределяет уже существующее свидетельство, на которое ссылается поле `source.adversarial-review-ref`, а не вводит новый тип артефакта требований.

7.4.6.1 Идентичность и множественность

Идентификатор — `AR-NNN`, сквозной в рамках родительского ТЗ и неизменяемый после создания (зеркально `ADAPT-NNN`, §7.4.1.4). Год несёт родительское ТЗ (`TZ-YYYY-NNN`); для delta-ТЗ нумерация ведётся в области действия соответствующего delta-ТЗ (§7.11).

Кардинальность — **ноль или более** AR на одно ТЗ, как и у ADAPT. Обзор при импорте ТЗ обязателен всегда и даёт первую AR; на стадиях деривации AR выпускается по факту проведения обзора. Единицей гранулярности является **событие обзора**: несколько артефактов, выведенных из одного обзорного события без новых вопросов, ссылаются на **одну** AR.

Полнота обеспечивается не мандатом «AR на каждом узле», а инвариантом провенанса узла:

> *Каждый BR / SR / SPEC ссылается ровно на один источник адаптации — `source.adapt` **либо** `source.adversarial-review-ref`. Состязательный обзор обязателен при импорте ТЗ; на стадиях деривации его исход фиксируется как AR по факту проведения.*

7.4.6.2 Обязательные поля

Поле	Обязательность	Значение
`id`	обязательно	`AR-NNN`; неизменяем
`tz-ref`	обязательно	Обозреваемое (delta-)ТЗ
`trigger-stage`	обязательно	Стадия обзора: `import-tz` / `decompose-br` / `decompose-sr` / `spec` / `tc` (перечень согласован с `trigger-stage` ADAPT, §7.8.1)
`reviewer.vendor` + `reviewer.model`	обязательно	Модель состязательного рецензента; **обязана отличаться** от модели основного агента (§7.10.2)
`verdict`	обязательно	`findings-present` либо `no-findings`
`produces-adapt[]`	условно	Обязательно непустой при `verdict: findings-present`; отсутствует либо пуст при `no-findings`
`status`	обязательно	`draft` / `issued` / `superseded`
`superseded-by`	условно	Обязательно при `status: superseded`
`signature.author` + `signature.timestamp`	обязательно для `issued`	Возможность носителя V6

При исходе `findings-present` тело AR содержит только указатели на разделы ТЗ и на порождённые записи `B-NNN`; **сами находки живут в ADAPT и в AR не дублируются**. При исходе `no-findings` тело содержит краткое обоснование однозначности конвертации.

7.4.6.3 Жизненный цикл

```
draft → issued → superseded
```

Статус	Что значит
`draft`	Обзор проведён, вердикт оформляется; ссылаться на AR из `source.adversarial-review-ref` **запрещено**
`issued`	Вердикт выпущен и подписан рецензентом (V6); запись **неизменяема**
`superseded`	Более поздний обзор на той же стадии заместил ранний вердикт; терминальное состояние, запись сохраняется для аудита (V1)

Выпущенная (`issued`) AR не редактируется. Единственный выход — `superseded`, когда повторный обзор на той же стадии выносит новый вердикт: тогда выпускается новая AR, а прежняя получает `status: superseded` и `superseded-by`. Модель замещения — та же, что для ADAPT (§7.6.4); отдельного контрольного гейта для AR не вводится (§10.11.1).

Висячая ссылка `source.adversarial-review-ref` на несуществующую AR либо на AR в статусе `draft` или `superseded` — **fatal**: гейт блокирует изменение (§10.11.1).

7.5 Утверждение ADAPT — подпись архитектора

ADAPT — **внутренний артефакт интерпретации**. Его аудитория — инженер, AI-агент, состязательный рецензент и верификатор; клиент его не видит и не подписывает. Обязательства перед клиентом живут в ACTZ (§7.13).

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

Подпись	Кто	Что подтверждает
Подпись архитектора	Архитектор со стороны исполнителя	Все обратные находки отработаны (каждая — в статусе `resolved` и несёт `decided-in` на пункт **подписанного** ACTZ, §7.4.5); прямая интерпретация технически реализуема и соответствует принятым решениям

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).

Почему подпись клиента здесь отсутствует. Клиентская подпись под ADAPT была фикцией: клиент подписывал инженерный документ, который не читал и не мог оценить. Граница проводится **по аудитории**: всё, что показано клиенту и утверждено, — обязательство (ACTZ, двусторонняя подпись); всё, что не показано, — интерпретация (ADAPT, подпись архитектора). Содержание записи при этом не обсуждается: спорить о том, «толкование это или изменение», больше не нужно — достаточно спросить, выносилось ли решение клиенту.

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.

Errata, не затрагивающая ни одного решения из подписанного ACTZ (исправление ошибки толкования, не влияющей на обязательства перед клиентом), клиента **не касается** и подписывается только архитектором (§7.6.3).

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-T3. Backward findings фиксируются по delta-T3. Approval — подпись архитектора §7.5; решения клиента по находкам delta-T3 фиксируются подписанными ACTZ (§7.13).
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-T3, 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 инженерный 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. Если правка затрагивает решение из подписанного ACTZ (меняет обязательство) — требуется **новый ACTZ** с двусторонней подписью; если не затрагивает — достаточно подписи архитектора

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 зафиксировано пунктом подписанного ACTZ) — дезавуирование требует **нового ACTZ с двусторонней подписью**: согласованное с клиентом решение нельзя отменить в одностороннем порядке, и отменяющий протокол проставляет `superseded-by` на прежнем (§7.13.4). Если правка не затрагивает ни одного решения из подписанного ACTZ, достаточно подписи архитектора (по аналогии с правилом errata, §7.6.3). Отдельный QG не вводится — дезавуирование проходит через тот же QG-3 (§7.5, §10.8.5).
3. **Состояние.** Дезавуируемый `ADAPT-MMM` переходит в выделенное терминальное состояние `superseded` — отдельное от `frozen` и от `obsolete` , которым устаревают TR / SPEC / TC (у ADAPT состояния `obsolete` нет, §10.8.1). `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`

```
# 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 TZ
```

Поле `source.adapt` — условное: присутствует, когда конвертация TZ → 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 TZ, (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 (§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 | asked | answered | approved | frozen | superseded
created: "<ISO-date>"
last-updated: "<ISO-date>"

approval:                               # обязательно для approved
  # client-signature ОТСУТСТВУЕТ: ADAPT — внутренняя интерпретация (§7.5).
  # Клиентские обязательства несёт ACTZ (§7.13) с двусторонней подписью.
  architect-signature:                  # mandatory для approved
    signed-by: "<name>"
    role: architect
    signed-at: "<ISO-datetime>"
```

```yaml
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 — архитектор
вычитал текст (§7.5)
---
```

Примечание: ADAPT существует только в одном режиме (подпись архитектора, 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 архитектором и состязательным рецензентом (§7.5).
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 §10.8. 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-asked	Все обратные находки в `asked-to-client`; вопросы вынесены клиенту в составе ACTZ (`status: sent`)	Ожидание подписания протокола

QG-ADAPT-answered	Все обратные находки в `answered` ; начато разрешение	Готов к финализации
QG-ADAPT-approve	Все обратные находки в `resolved` и несут `decided-in` на подписанный ACTZ; подпись архитектора 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 вердикте состязательного рецензента «findings present» (§7.4.1) AI-агент создаёт **draft ADAPT**: 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 §3.5): 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
  ar/
    AR-001.md
    AR-002.md
  tz/
    TZ-YYYY-NNN.md
    TZ-YYYY-NNN-delta-1.md
```

Записи состязательного обзора (§7.4.6) хранятся в подпапке `ar/`. Они существуют и тогда, когда ADAPT не создан (вердикт `no-findings`), — поэтому подпапка независима от `adapt/`.

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 (подпись архитектора §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-ТЗ — 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-ТЗ через `source.tz-section`.

7.12.4 Связь с инверсией источника истины

ТЗ — договорной артефакт, но **не** источник истины о поведении системы (§2.3). Источник истины — RENAR-описание (BR / SR / SPEC / TR / TC). ТЗ — 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 (ТЗ) → источник истины (RENAR-описание) → код. ADAPT встраивается в цепочку реактивно, когда мост между языками нужен.

7.13 ACTZ — протокол уточнения ТЗ

7.13.1 Назначение и граница с ADAPT

ACTZ (`ACTZ-NNN` , Agreed Clarification of TZ, в договорном обороте — «**Протокол уточнения ТЗ № N**») — артефакт **контрактного контура**: порция вопросов, предложений и решений, вынесенная клиенту и подписанная обеими сторонами.

Граница между ACTZ и ADAPT проводится **по аудитории**, а не по содержанию записи:

> ***Всё, что показано клиенту и утверждено, — обязательство. Всё, что не показано, — интерпретация.***

	ADAPT — интерпретация	**ACTZ — утверждение**
Содержимое	Перевод языка ТЗ на инженерный, сопоставление терминов, достроенные сценарии, обратные находки, прямая интерпретация	Вопросы, предложения и решения, вынесенные клиенту. Формулируются на языке обязательств («кнопка называется X»), а не толкования
Аудитория	Инженер, AI-агент, состязательный рецензент, верификатор	Клиент и стороны договора
Подпись	Только архитектора (§7.5)	**Двусторонняя**: клиент + исполнитель
Вес	Внутренний контур	**Контрактный**

Критерий бинарен и проверяем: факт «вынесено клиенту и подписано» либо есть, либо нет. Спор о существенности правки («это уточнение или уже изменение?») исчезает по построению.

7.13.2 Происхождение и множественность

ACTZ порождается двумя способами, оба штатны:

1. **Из обратной находки.** Находка, требующая решения клиента (§7.4.4), выносится в ACTZ; после подписания запись получает `decided-in: ACTZ-NNN §M` (§7.4.5).
2. **По инициативе клиента, без находки.** Клиент сам предлагает решение («кнопку назовём иначе»). Такой ACTZ ссылается только на ТЗ; родительской находки у него нет. После подписания решение **обязано** быть отражено в ADAPT — интерпретация реагирует на протокол.

Кардинальность:

Отношение	**Кардинальность**	**Пояснение**
ТЗ : ACTZ	**1 : 0..N**	Ноль протоколов законен: конвертация без вопросов и без инициатив клиента не порождает документов
ADAPT : ACTZ	**1 : 1..N**	Находки одного ADAPT закрываются несколькими протоколами по мере раундов уточнений

Поздний ACTZ — штатный случай, а не исключение. Протокол возникает на **любой стадии жизни заказа**, в том числе из разбора замечаний заказчика после демонстрации. Норма стадийно-независима — симметрично реактивному ADAPT (§7.4.1).

7.13.3 Обязательные поля

Поле	Обязательность	Значение
`id`	обязательно	`ACTZ-NNN`; сквозной в рамках родительского ТЗ; неизменяем
`tz-ref`	обязательно	ТЗ, к которому относится протокол
`resolves[]`	условно	Записи `B-NNN` в ADAPT, которые закрывает протокол; отсутствует у ACTZ по инициативе клиента
`decisions[]`	обязательно	Пункты решений; каждый формулируется на языке обязательств и имеет стабильный номер `§M`
`annexes[]`	условно	Новые версии приложений к ТЗ, утверждаемые этим протоколом (§7.13.5)
`status`	обязательно	`draft` → `sent` → `signed` → `superseded`
`client-signature`	обязательно для `signed`	Клиент или его представитель с полномочиями (V6: author + timestamp)
`vendor-signature`	обязательно для `signed`	Исполнитель (V6)
`superseded-by`	условно	Обязательно при `status: superseded`

7.13.4 Жизненный цикл

```
draft → sent → signed → superseded
```

Статус	Что значит
`draft`	Протокол готовится; клиенту не вынесен. Ссылаться на него из `decided-in` **запрещено**
`sent`	Вынесен клиенту, ожидается подписание
`signed`	Подписан обеими сторонами; **неизменяем**, несёт контрактный вес
`superseded`	Более поздний подписанный ACTZ отменил решение; терминальное состояние, запись сохраняется для аудита (V1)

Подписанный ACTZ **не редактируется**. Исправление — только новым ACTZ, отменяющим прежнее решение (`superseded-by`); модель замещения — та же, что для ADAPT (§7.6.4).

Клиенту выносится **подготовленный список решений**, а не сырой вывод AI-агента (§7.10.3): вопросы агрегирует и переформулирует архитектор.

7.13.5 Приложения к ТЗ

Приложения (карты сопоставления, справочники, макеты) — **неотъемлемая часть ТЗ**, отдельной адресуемой сущностью они не являются. Уточнение приложения оформляется **новой версией**

приложения, приложенной к ACTZ и утверждаемой той же двусторонней подписью (`annexes[]`).

Это закрывает случай, когда решением клиента является не текст, а новая версия таблицы: прежняя версия остаётся неизменяемой (V1), новая входит в итоговое ТЗ через подписавший её протокол.

7.14 Итоговое ТЗ — эталон сдачи-приёмки

Итоговое ТЗ (effective TZ) — эталон, против которого система сдаётся и принимается:

> ***Итоговое ТЗ = начальное ТЗ (включая приложения) + все подписанные ACTZ.*** *При расхождении между документами приоритет имеет* ***более поздний подписанный*** *документ.*

Против итогового ТЗ выводятся приёмочные тесты АТ (§9.19); из него же строится отчёт приёмки.

ADAPT в эталон приёмки не входит. Внутренняя интерпретация из контрактного контура выведена целиком — клиент отвечает только за то, что подписывал и понимал. Это и есть смысл расщепления: приёмка становится юридически чистой.

QG-4 (§10.4.2) — отдельный **опциональный** гейт бизнес-результата (`achievement ≥ 80%`). Он **не подменяет** приёмку соответствия итоговому ТЗ и с ней не смешивается: бизнес-цель может быть достигнута при несоответствии контракту, и наоборот.

7.15 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	Подпись архитектора на ADAPT и двусторонняя подпись ACTZ (V6) + atomic approval (V2 + V3); immutability после approved (V1); delta-ADAPT через V4
13 Соответствие	Состязательный обзор для каждого ТЗ — обязательное положение; ADAPT создаётся реактивно (§7.4.1)

08. Спецификации — 11 типов SPEC

> ***Часть RENAR Standard v1.0*** · *← 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 одиннадцать, и список закрыт: `SPEC-ARCH` , `API` , `DATA` , `INT` , `PROC` , `UI` , `AI` , `SEC` , `OPS` , `TEST` , `DOC` — новый тип вводится только через формальную процедуру изменения стандарта (глава 13).

Спецификация описывает **три разных предмета**, и путать их нельзя. Девять типов описывают **устройство системы**. `SPEC-TEST` описывает **конструкцию доказательства** её соответствия: стенд интеграционной системы — не инсталляция, а инженерное изделие (эмуляторы, датасеты по обе стороны каждой интеграции, правила сверки с данными чужих систем). `SPEC-DOC` описывает **состав поставляемого результата**: документация, которую заказчик получает как часть поставки.

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
Доказательная	`SPEC-TEST`	На чём и на каких данных доказывается соответствие (§8.3.2)
Поставочная	`SPEC-DOC`	Что входит в поставку заказчику помимо самой системы

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     SPEC-TEST
                                             SPEC-DOC

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 одиннадцати 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
`SPEC-TEST`	Тестовые стенды и данные: топология, эмуляторы и sandbox-инстансы внешних систем, конфигурация прогона (что мокается, что живое), датасеты по обе стороны каждой интеграции, правила сверки результатов с данными чужих систем, объём и анонимизация	ISO/IEC/IEEE 29119-4 (test techniques), Testcontainers, Pact
`SPEC-DOC`	Поставляемая проектная документация: состав (руководство пользователя, руководство администратора, обучающие материалы), структура каждого документа (обязательные разделы), проверяемые требования к нему (полнота по ролям и сценариям, актуальность версии, формат)	ISO/IEC/IEEE 26511, ISO/IEC/IEEE 26514

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~~	**Решение отменено в v1.0**	Прежнее решение («тестовые окружения — раздел SPEC-OPS») отменяется: см. §8.3.2. Тестовый стенд введён отдельным типом SPEC-TEST
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

Отменённое решение сохраняется в таблице зачёркнутым, а не вычёркивается: история решений — часть стандарта, и читатель вправе видеть, что и почему было пересмотрено.

8.3.2 Ревизия решения по тестовым стендам

Прежнее решение отождествляло тестовый стенд с **окружением развёртывания** и отправляло его в SPEC-OPS (поле environments[] : dev / staging / prod). Для простой системы это верно: стенд там — стейдж плюс датасет. Решение отменено по трём причинам.

1. Стенд интеграционно-тяжёлой системы — другой объект. Это не инсталляция системы, а **конструкция доказательства**: эмуляторы и sandbox-инстансы внешних систем; согласованные датасеты **по обе стороны** каждой интеграции; правила сверки результатов с данными чужих систем — ожидаемый результат живёт **не в нашей системе**; конфигурация прогона (что мокается, что живое). Стандарт это уже чувствовал: §9.6.3 требует для SPEC-INT прогон «против реальной или sandbox-counterparty — мокированного контракта недостаточно», но носителя для описания этой counterparty не давал.

2. Ложные инвалидации — решающий довод. §10.5.4 инвалидирует verified при **любом** инкременте версии артефакта, автоматически и без разбора причины. Если бы TC ссылался на стенд через SPEC-OPS, то правка **процедуры деплоя** — к стенду отношения не имеющая — обрушивала бы в approved **все** TC, ссылающиеся на этот SPEC-OPS. Деплой правят чаще стенда, и доказательная база обесценивалась бы регулярно и ложно. Отдельный SPEC-TEST делает инвалидацию **точной**: TC инвалидируются тогда и только тогда, когда изменилось то, против чего они валидны.

3. Чужеродная подпись. Чувствительные тестовые данные (объём, анонимизация, использование реальных данных клиента) согласует **клиент**. Клиентская подпись на разделе эксплуатационного документа чужеродна; на отдельном артефакте — естественна.

Границы новых типов:

Тип	Отвечает на вопрос
SPEC-OPS	**Как система живёт** в эксплуатации (деплой, наблюдаемость, SLO, runbook)
SPEC-TEST	**Как доказывается** её соответствие (стенды, данные, конфигурация прогона)
SPEC-DOC	**Что входит в поставку** заказчику (документы как предмет договора)

Граница SPEC-DOC ↔ SPEC-OPS. Критерий — **вхождение в состав поставляемого результата**, а состав задаётся договором **до** приёмки, а не выясняется на ней:

- **Руководство администратора — всегда** `SPEC-DOC` : если систему эксплуатирует заказчик, руководство для его администраторов входит в состав поставки. «Внутренней версии» этого жанра не существует.
- **Runbook — всегда** `SPEC-OPS` : внутренний операционный регламент эксплуатирующей команды исполнителя; заказчику не предъявляется, предметом приёмки не является.

Двойная принадлежность документа исключена по построению — ребро `TR.implements-spec[]` всегда однозначно.

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 11 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.

```
---
# === Identity (обязательно) ===
id: SPEC-<TYPE>-NN[.N]              # immutable; TYPE ∈
{ARCH,API,DATA,INT,PROC,UI,AI,SEC,OPS,TEST,DOC}
title: "<short, descriptive>"
type: SPEC-ARCH | SPEC-API | SPEC-DATA | SPEC-INT | SPEC-PROC | SPEC-UI | SPEC-AI
| SPEC-SEC | SPEC-OPS | SPEC-TEST | SPEC-DOC
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: AR-NNN   # обязательно если adapt omitted (§7.4.6)

# === Граф связей (auto-managed except mandatory ones) ===
referenced-by: []                   # auto-derived; SR/TR/SPEC ссылающиеся сюда
```

```yaml
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).

8.5.10 SPEC-TEST

Type-specific frontmatter: `benches[]` (топология стенда: узлы, версии, что живое / что эмулируется), `counterparties[]` (эмуляторы и sandbox-инстансы внешних систем — по одной записи на интеграцию), `datasets[]` (датасет с обеих сторон интеграции: объём, происхождение, `anonymized: boolean`, `contains-real-client-data: boolean`), `reconciliation-rules[]` (правила сверки результатов с данными чужих систем), `run-config` (что мокается, что живое, порядок прогона), `client-signature` (**обязательна**, если хотя бы один датасет несёт реальные или персональные данные клиента).

Обязательное тело: топология стенда; перечень внешних систем и способ их представления (реальная / sandbox / эмулятор — с обоснованием); датасеты по обе стороны каждой интеграции; правила сверки (**где живёт ожидаемый результат**, если он не в нашей системе); конфигурация прогона; политика данных (объём, анонимизация, срок хранения).

Spec-specific TC: воспроизводимость (повторный прогон на том же стенде даёт тот же результат); валидность counterparty (sandbox отвечает как реальная система — иначе доказательство фиктивно); целостность датасета (данные соответствуют объявленной схеме и объёму).

Связь с TC. Каждый TC с `automation.kind: dynamic` обязан нести `environment-ref` на SPEC-TEST (§9.3): «тест пройден» имеет смысл только против конкретного стенда и конкретных данных. Статические проверки (док-линт §9.8, структурный TC §9.8.1) стенда не требуют — анализатор работает по артефактам, — и поле к ним не применяется. Смена стенда или датасета инкрементит версию SPEC-TEST и по §10.5.4 инвалидирует `verified` у зависимых TC — **точечно**, не задевая всё остальное.

Подпись клиента. Объём, анонимизация и допустимость использования реальных данных клиента — предмет **его** решения, а не исполнителя. При `contains-real-client-data: true` или неполной анонимизации `client-signature` обязательна.

8.5.11 SPEC-DOC

Type-specific frontmatter: `deliverables[]` (по одной записи на документ: `kind` — руководство пользователя / руководство администратора / обучающие материалы / иное; `audience`; `format`; `language`), `required-sections[]` (обязательные разделы каждого документа — **задаются как требование**, а не оставляются на усмотрение исполнителя), `traces-to[]` (BR / SR / SPEC, поведение которых документ описывает), `version-binding` (правило соответствия версии документа версии системы), `acceptance-criteria[]` (по чему документ принимается).

Обязательное тело: состав поставки (какие документы и для какой аудитории); структура каждого документа; требования к нему (полнота по ролям и сценариям, заявленным в требованиях; актуальность версии; язык и формат); критерии готовности.

Spec-specific TC: **док-линт** (§9.8) — обязателен.

Почему док-линт обязателен. Неверифицируемый SPEC не проходит QG-2 (§9.7, MVR-5 требует покрытия каждого нормативного утверждения). SPEC-DOC без машинной проверки либо заблокировал бы поставку, либо вынудил проделать дыру в MVR-5 ради одного типа. Поэтому его утверждения нормируются проверяемо: наличие обязательных разделов, покрытие всех ролей и сценариев из связанных BR / SR, совпадение версии документа с версией системы, формат и язык — всё это проверяется машинно (`automation.kind: static`). Содержательное соответствие реализованному поведению («текст не врёт о системе») проверяется опционально judge-агентом через уже существующий механизм P7 / `eval` — новых сущностей не требуется.

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[]` :

```yaml
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 11 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 §10.7):

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
    test/     SPEC-TEST-NN-*.md
    doc/      SPEC-DOC-NN-*.md
  adapt/
  tz/
  SPECS.md                # auto-generated index
```

> *Все 11 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
    test/     SPEC-TEST-NN-*.md
    doc/      SPEC-DOC-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 11 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*** · *← 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 business goal| H[tc-type: business]
  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 §3.5; 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).
P8	Изоляция авторства теста	Тест отделён от реализации по трём осям: **время** — тест заморожен (`ready`) до начала реализации, которую он проверяет; **автор** — агент, пишущий тест, не совпадает с агентом, пишущим реализацию (§9.18); **изменение** — правка критериев замороженного теста только через `[test-spec-change]` с утверждением инженера (§9.13).
P9	Красная история	Тест, ни разу не наблюдавшийся красным, **не является доказательством**. Фиксирующий прогон выполняется до реализации: тест обязан быть красным; переход «красный → зелёный» записывается (§9.18).

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

Зачем P9, если есть P8. Изоляция авторства гарантирует, что тест написан **до** кода и **не тем**, кто пишет код. Но она не гарантирует, что тест **что-то проверяет**: пустой тест, честно написанный изолированным агентом по ошибке, формально безупречен и зелен с рождения — он проходит все три оси P8. Красная история закрывает ровно этот остаток, и больше ничто его не закрывает.

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

# === Classification (mandatory) ===
tc-type: business | 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

# === Task binding (optional; §9.19.7) ===
verifies-tr: TR-NN                  # задача, к объёму которой сужен тест
verifies-claims: []                 # подмножество утверждений родительского SR,
покрываемое в объёме TR

# === Красная история (mandatory для засчёта TC как доказательства; §9.18.2) ===
red-history:
  fixing-run: { date: "<ISO-8601>", result: fail }     # фиксирующий прогон ДО
реализации — обязан быть красным
```

```yaml
  green-transition: "<ISO-8601>"                          # момент перехода красный → зелёный (ведёт runner)
  inherited-from: TC-NN                                   # conditional: задача без изменения поведения (рефакторинг)
  not-applicable-reason: implementation-originated        # conditional: класс §6.13; требует mutation-check ниже
mutation-check:                                           # mandatory если not-applicable-reason задан (§6.13.3)
  mutants-killed: integer                                 # обязан быть ≥ 1

# === Стенд и данные (conditional; §8.5.10) ===
environment-ref: SPEC-TEST-NN        # mandatory если automation.kind: dynamic — стенд и датасет,
                                     # против которых TC валиден. Для static (док-линт §9.8,
                                     # структурный TC §9.8.1) не применяется: анализатор работает
                                     # по артефактам, а не против стенда

# === Automation (mandatory) ===
automation:
  status: automated | manual-pending
  kind: dynamic | static             # mandatory; static — runner является статическим анализатором (§9.8.1)
  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 ∈ { business, ux, system, contract, eval, security }
```

Тип	Что проверяет	Применяется к	runner-семейство
`business`	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.

Почему `business`, а не `acceptance`. Приёмок в RENAR две, по разные стороны развязки контуров: `business`-TC проверяет достижение бизнес-цели BR (внутренний контур), **AT** (§9.19) — соответствие контракту, то есть итоговому ТЗ. Одно имя на два разных предмета — прямой источник путаницы; имена разведены.

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)
SPEC-TEST	Воспроизводимость (повторный прогон на том же стенде даёт тот же результат); Валидность counterparty (sandbox отвечает как реальная система); Целостность датасета (данные соответствуют объявленной схеме и объёму)	Дрейф стенда (конфигурация не разошлась с объявленной); Полнота анонимизации
SPEC-DOC	**Док-линт** (`automation.kind: static`): наличие обязательных разделов; покрытие всех ролей и сценариев из связанных BR / SR; совпадение версии документа с версией системы; язык и формат	Содержательное соответствие реализованному поведению — judge-агентом через P7 / `eval`

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.

Самореференция SPEC-TEST. TC, верифицирующие сам `SPEC-TEST` (воспроизводимость, валидность counterparty, целостность датасета), прогоняются **на том же стенде**, который они описывают, и потому несут `environment-ref` на него же. Круг допустим и является нормой: стенд доказывает собственную пригодность единственным доступным способом — работая. Инвалидация при этом остаётся корректной: смена стенда инкрементит его версию и обесценивает в том числе его собственные TC — что и требуется.

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

SPEC-DOC — док-линт (обязателен). Проверяемые утверждения поставляемой документации (§8.5.11): наличие обязательных разделов; покрытие всех ролей и сценариев, заявленных в связанных BR / SR; совпадение версии документа с версией системы; язык и формат поставки. Runner — статический анализатор (`automation.kind: static`). Содержательное соответствие реализованному поведению проверяется опционально judge-агентом через P7 / `eval` — отдельного механизма не вводится.

Без док-линта SPEC-DOC неверифицируем: QG-2 требует покрытия каждого нормативного утверждения (§9.7), и такой SPEC либо заблокировал бы поставку, либо вынудил ослабить MVR-5 ради одного типа.

SPEC-TEST — воспроизводимость, валидность counterparty, целостность датасета (§8.5.10). Валидность counterparty — не формальность: sandbox, отвечающий не как реальная система, превращает доказательство в фикцию.

9.8.1 Структурный TC — вырожденная нормативная часть

Для SPEC-ARCH и SPEC-DATA обязательный вид TC — соответствие структуры (правила зонирования и зависимостей, соответствие схем). Его runner — **статический анализатор**, а не прогон системы; это фиксируется полем `automation.kind: static` (§9.3). Отдельного типа TC для структурных проверок **не вводится**: проверка уже обязательна, недоставало лишь признания природы её runner'а.

> *У структурного TC нормативная часть **вырождена**: утверждение уже записано в SPEC-ARCH / SPEC-DATA; нормативная часть теста лишь указывает, какое положение спецификации подлежит автоматической проверке.*

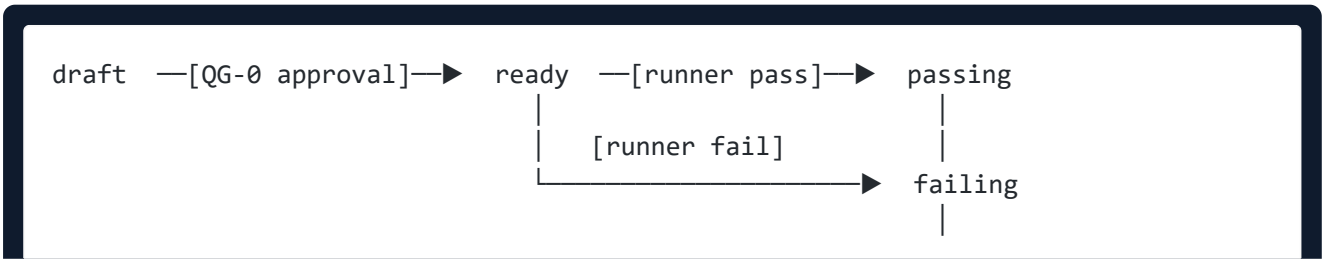
Из вырожденности следует **маршрут разбора провала** — иначе неизбежен спор «что чинить: код, тест или спецификацию?»:

Причина провала	Что меняется
Код нарушает правило, записанное в SPEC	**Код**
Правило признано неверным либо требует ужесточения	**SPEC** — новая версия (не правка теста!)
Анализатор даёт ложный результат	**Инструмент**; эталон не тронут

Правка структурного TC ради «зеленения» провала — подгонка теста под код в чистом виде: эталон живёт в SPEC, и менять надо его.

9.9 Жизненный цикл TC

9.9.1 Машина состояний



```
            [delta-T3 invalidation;                          |
             see §9.16]                                      |
                  ┌──────────────────────────────────────────┘
                  ▼
              obsolete
```

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	TC (`draft → ready`) — часть	`automation.status = automated` (или `manual-pending` с дедлайном	TC переходит в `ready` ; допускается

проверки (§10.3.2)	«реализация»	и причиной); pos/neg парность (§9.7); dry-run runner прошёл; обязательные секции body TC (§9.4) заполнены	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

Выборка **наводится машинными сигналами**, а не берётся случайно. После введения красной истории (§9.18.2) слепая выборка избыточна: носитель знает, какие тесты подозрительны.

Параметр	Нормативное требование
Размер выборки	5 TC (по умолчанию); может быть увеличено в манифесте соответствия проекта
Приоритет отбора	1) TC **без красной истории** (фиксирующий прогон не наблюдался красным); 2) TC, **переживший мутанта** (мутационная проверка не убила ни одного мутанта), если носитель её выполняет; 3) TC класса `implementation-originated` (§9.18.2); 4) остальные — случайным добором до размера выборки
Распределение по типам	Равномерное по `tc-type` в пределах случайного добора
Статус	Только `passing`
Кто выбирает	AI-агент по приоритету выше; случайный добор — нативной для носителя случайностью (seed фиксируется в audit-feed)

Случайная выборка ловит слабый тест с вероятностью, обратной размеру набора; направляемая — предъявляет инженеру ровно те тесты, у которых **машина уже нашла признак незубастости**. Дешевле и точнее.

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:

```yaml
last-spot-check:
  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

Две метрики покрытия считаются в **разных единицах**, и подменять одну другой запрещено. `coverage-percent` измеряет **продвижение артефактов** (сколько артефактов дошло до `verified`) — единица счёта — артефакт. Норма pos/neg-парности (§9.7, MVR-5) измеряет **полноту верификации** — единица счёта — нормативное утверждение, и цель здесь всегда 100%, без порога в манифесте. Артефакт с зелёным `coverage-percent` может содержать утверждение, покрытое только положительным TC; именно поэтому QG-2 блокирует promote по утверждениям (§10.5.2), а не по факту наличия одного негативного TC на артефакт.

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-T3 и TC

9.16.1 Impact analysis по тестам

При delta-T3 (глава 7 §7.6) AI-агент выполняет impact analysis по TC 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/
    business/    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 Изоляция авторства теста и красная история

9.18.1 Три оси изоляции (P8)

Тест, созданный в процессе написания кода, агент подстроит **под код**, а не под требование: он честно проверит то, что написано, — но не то, что требовалось. Существующая защита (§9.13) запрещает менять код и критерии **существующего** TC в одной атомарной единице изменения, но не запрещает написать **новый** TC после кода. Изоляция закрывает это на трёх осях.

Ось	Нормативное требование	Проверка
Время	TC замораживается (`ready`) **до** начала реализации, которую он проверяет. TR не переходит в работу, пока связанные с ним TC не в `ready`	hook носителя блокирует старт TR
Автор	Агент, создающий TC, **не совпадает** с агентом, создающим реализацию (разные сессии или модели). Расширение P7 с `ux` / `eval` на все типы TC	Провенанс TC (`ai-provenance.generated-by`) сверяется с провенансом change unit реализации

Изменение	Правка Pass/Fail-критериев замороженного теста — только через `[test-spec-change]` с утверждением инженера (§9.13.2)	hook носителя

9.18.2 Красная история (P9)

> ***Тест, ни разу не наблюдавшийся красным, не является доказательством.***

Фиксирующий прогон выполняется **до** реализации: тест обязан быть **красным** — проверяемой функциональности ещё нет. Переход «красный → зелёный» записывается носителем как часть истории прогонов и является **условием**, при котором TC засчитывается как доказательство при QG-2.

Зелёный результат на фиксирующем прогоне — **сигнал разбора**, а не успех: либо тест ничего не проверяет, либо проверяемая функциональность уже существует. Оба случая требуют разбора инженером, и ни один не позволяет считать тест доказательством.

Наследование. У задач, не меняющих наблюдаемое поведение (рефакторинг, пересборка, миграция без изменения контракта), красная история **наследуется** от TC того же утверждения: требовать нового красного прогона там бессмысленно — поведение не менялось.

Исключение — тесты класса `implementation-originated` (§6.13). Такой тест пишется **после** кода и рождается зелёным: красной истории у него быть не может по построению. Компенсация обязательна: для этого класса TC **мутационная проверка обязательна** — тест, рождённый зелёным, доказывает свою работоспособность **убитым мутантом**. Без убитого мутанта такой TC доказательством не является.

9.18.3 Машинный сигнал ослабления нормы

> *Если после правки теста **падает ранее зелёная система** — норма изменилась **по факту**, как бы правку ни классифицировали.*

Сигнал машинно-проверяем и не зависит от того, честно ли агент пометил единицу изменения тегом `[test-spec-change]`. Срабатывание — блокирующее: единица изменения не проходит, пока правка не будет проведена как изменение нормы (с утверждением инженера) либо отменена.

Это парная защита к P8: изоляция закрывает подгонку теста под код, машинный сигнал — молчаливое ослабление теста под уже написанный код.

9.19 AT — приёмочный тест контрактного контура

9.19.1 Назначение

Прослеживаемость TC замкнута через **интерпретацию**: `TC → SR → ADAPT → T3`. Отсюда класс дефектов, который уровень TC не ловит **в принципе**: если интерпретация T3 неверна, все TC могут быть `passing` — система идеально соответствует **неверной** интерпретации, — и при этом провалить приёмку у заказчика.

AT (`AT-NN`) — приёмочный тест, выводимый **исключительно** из **итогового T3** (§7.14): начальное T3 с приложениями плюс все подписанные ACTZ. AT проверяет соответствие **контракту**, а не

интерпретации.

9.19.2 Изоляция как механизм генерации

AT создаёт **изолированный агент**: на вход подаётся **только** итоговое ТЗ. Доступ к ADAPT, BR / SR / SPEC, TC и коду **запрещён**. Модель агента-генератора AT обязана отличаться от модели основного агента (зеркально P7).

Изоляция здесь — не гигиена, а **суть механизма**: AT есть симуляция взгляда заказчика. Агент, видевший ADAPT или SR, воспроизведёт **ту же** интерпретацию — и AT начнёт подтверждать её вместо контракта. Уровень приёмки схлопнется в уровень верификации, а ошибка толкования получит **ложное подтверждение соответствия**. Нарушение изоляции — **fatal** (§10.11.1).

9.19.3 Обязательные поля и тело

Поле	Обязательность	Значение
`id`	обязательно	`AT-NN`; неизменяем
`verifies[]`	обязательно	Только `TZ §N` и `ACTZ-NNN §M`. Ссылка на внутренний артефакт (BR / SR / SPEC / TC) — **fatal**
`tz-version`	обязательно	Редакция итогового ТЗ, из которой AT выведен (§9.19.4)
`generator`	обязательно	Провенанс изолированного агента: вендор, модель, подтверждение отсутствия доступа к внутреннему контуру
`negative`	обязательно	Pos/neg-парность — как у TC (§9.7)
`status`	обязательно	`draft → ready → passing / failing → obsolete` — та же машина состояний, что у TC (§9.9)
`environment-ref`	обязательно	Стенд и датасет приёмочных испытаний (§8.5.10). **Поле заполняется не генератором**: изолированный агент внутренних артефактов не видит и видеть не должен (§9.19.2). Привязку к стенду проставляет архитектор или runner **после** генерации — так испытания остаются воспроизводимыми, а изоляция не нарушается
`automation`	обязательно	Как у TC, включая обязательное `automation.kind` (§9.3): прогон только автоматическим runner'ом
`last-run`	ведёт runner	Как у TC (§9.12)

`tz_text` — дословная цитата проверяемого пункта итогового ТЗ рядом с шагами проверки — **обязательный** раздел тела AT. Цитата снимает спор на приёмке: предъявляется не пересказ, а сам пункт контракта.

9.19.4 Перегенерация перед испытаниями

> ***AT перегенерируются перед каждыми испытаниями** от **действующей** редакции итогового ТЗ. Программа испытаний обязана соответствовать той редакции контракта, против которой система предъявляется; устаревшая программа к испытаниям **не допускается**.*

Итоговое ТЗ живёт инкрементно: каждый подписанный ACTZ порождает новую редакцию. AT, выведенные при планировании, к моменту испытаний устаревают — и без перегенерации система в конце длинного заказа проверяется против контракта годичной давности. Перегенерация дёшева: её выполняет тот же изолированный агент.

Свежесть AT относительно действующей редакции итогового ТЗ проверяется гейтом; расхождение блокирует испытания.

9.19.5 Маршрутизация провалов

Расхождение уровней — это **диагностика**, а не шум:

AT	TC	Диагноз	Что чинится
✗	✓	**Ошибка интерпретации**: система соответствует ADAPT, но не контракту	ADAPT / ACTZ, затем производные требования
✗	✗	Дефект реализации	Код
✓	✗	Требование строже контракта либо TC неверен	Разбор: избыточное требование или дефект TC
✓	✓	Норма	—

Первая строка — то, ради чего вводится AT: этот дефект не обнаруживается никаким TC.

9.19.6 Отчёт приёмки

`ACCEPTANCE.md` — автогенерируемый отчёт (аналог `COVERAGE.md`, §9.15): покрытие AT по разделам итогового ТЗ и пунктам подписанных ACTZ. Продукт не предъявляется к сдаче, пока не все AT в статусе `passing` (§10.4.3).

9.19.7 Привязка теста к задаче (TR)

Приёмочные критерии TR уже нормированы (§6.7.2, §6.7.3) и проверяются QG-2 (§10.6.2); собственного класса артефактов TR не заводит. Пробел иной: критерии TR верифицируются тестами родительского SR, **которые шире объёма задачи**, и локальной проверки «сделана ли именно эта задача» не существует.

Вводятся поля TC:

Поле	Значение
`verifies-tr`	Задача (TR), к объёму которой привязан тест
`verifies-claims[]`	**Подмножество** нормативных утверждений родительского SR, покрываемое тестом в объёме этой задачи

Право сузиться — существо механизма. Привязанный к TR тест обязан иметь право проверять не весь SR, а ровно те его утверждения, которые входят в объём задачи. Без этого права поле бесполезно: тест снова оказывается шире задачи, и локальность не достигается.

Привязка **не ослабляет** покрытие: полнота по-прежнему считается от утверждений артефакта (§9.7), а не от задач. `verifies-tr` даёт *локальный* критерий готовности TR, а не альтернативный

способ закрыть SR.

9.20 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

> ***Часть RENAR Standard v1.0*** · *← Оглавление*
>
> ***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`, пока каждая находка не закрыта подписанным протоколом уточнения. 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 -->|draft → 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)	Подпись архитектора (ADAPT / SPEC-ARCH)	V3 + V6
Подписание ACTZ	Двусторонняя подпись (клиент + исполнитель)	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.
- **Красная история** (§9.18.2): каждый TC из `verified-by` имеет записанный переход «красный → зелёный». TC, ни разу не наблюдавшийся красным, доказательством не является и QG-2 не закрывает. Исключение — TC класса `implementation-originated` (§6.13): для него обязателен **убитый мутант**.
- 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 Gates за пределами обязательного ядра

Раздел описывает две разные вещи, и их нельзя смешивать.

Опциональные QG-3 и QG-4 (§10.4.1, §10.4.2) — 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.

Релизный гейт приёмки AT (§10.4.3) — **обязателен** и опциональным не является. Он **не входит** в закрытый список Quality Gates (§10.10.1) и не расширяет его: QG нормируют переход **артефакта** между состояниями, тогда как релизный гейт AT нормирует предъявление **продукта** к испытаниям и собственного `QG-N` не имеет.

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).
- Каждая обратная находка несёт `decided-in` на пункт **подписанного** ACTZ (глава 7 §7.13).
- Подпись архитектора готова (глава 7 §7.5). Подпись клиента на ADAPT **не требуется**: клиентские обязательства несёт ACTZ.
- 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 — выносится в `guide/`).

Постусловие:

- ADAPT переходит в `approved` (immutable наравне с T3).
- Запись в лог переходов с подписью архитектора (V6 author + timestamp).

Триггер: подпись архитектора на ADAPT (§7.5). Двусторонняя подпись стоит на ACTZ (§7.13) и фиксируется атомарно (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 Релизный гейт приёмки (AT)

Назначение: продукт не предъявляется к сдаче-приёмке, пока не доказано его соответствие **контракту**.

Предусловие:

- Все AT (§9.19) в статусе `passing`.
- AT выведены из **действующей** редакции итогового ТЗ (§7.14): `tz-version` каждого AT совпадает с текущей редакцией. Устаревшая программа испытаний к испытаниям **не допускается** (§9.19.4).
- Провенанс генератора AT подтверждает изоляцию: модель отлична от модели основного агента, доступ к внутреннему контуру отсутствовал (§9.19.2).

Постусловие: продукт может быть предъявлен к испытаниям; отчёт `ACCEPTANCE.md` актуален.

Триггер: изолированный агент-генератор AT + автоматический runner.

Гейт **обязателен** и не смешивается с QG-4 (§10.4.2): QG-4 опционален и меряет бизнес-результат, релизный гейт AT — соответствие контракту. Бизнес-цель может быть достигнута при несоответствии контракту, и наоборот.

10.4.4 Соответствие стандарту с опциональными гейтами

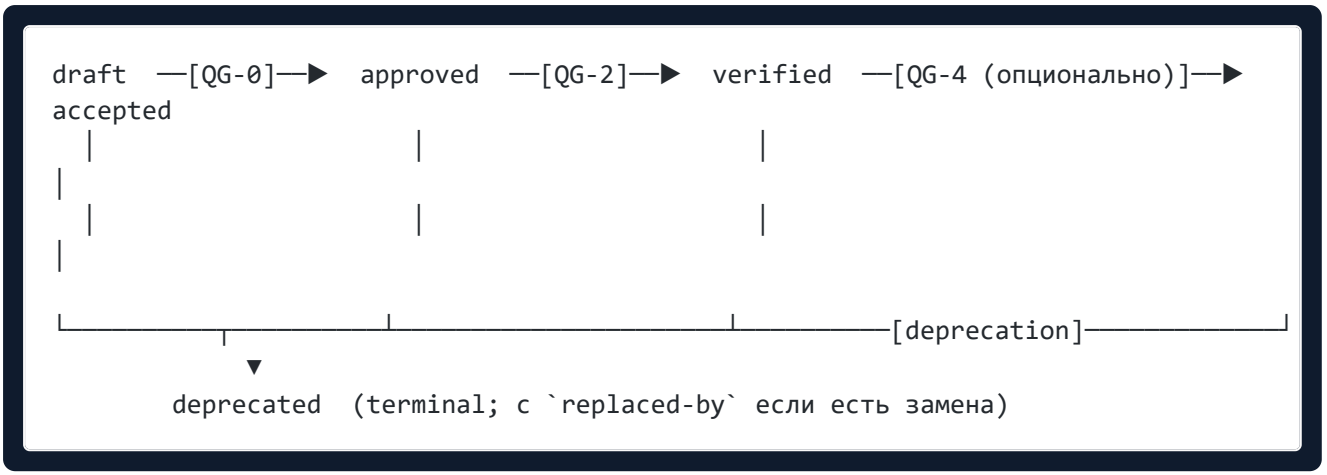
Манифест соответствия (глава 13) обязан явно объявлять:

```
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	Pos/neg парность **по каждому нормативному утверждению** артефакта (§9.7, MVR-5): каждое утверждение, описывающее наблюдаемое поведение, покрыто парой positive + negative TC. Единственное исключение — утверждение само описывает negative invariant. 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.

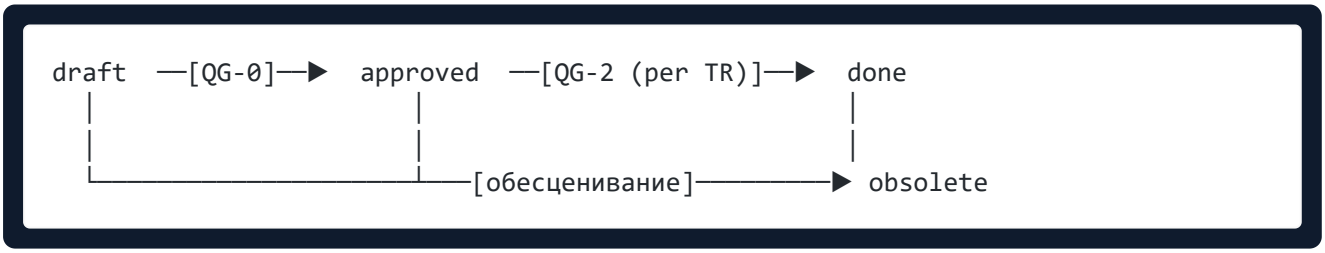
Триггер: архитектор или Владелец продукта.

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

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:

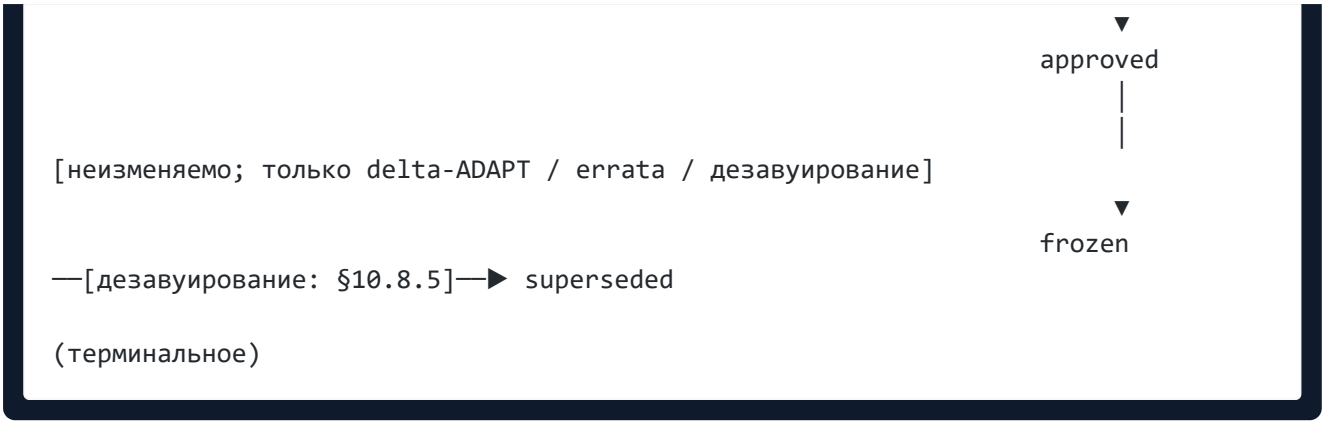
- `depends-on[]` граф ацикличен (глава 8 §8.6.3).
- Все SPEC из `depends-on[]` в состоянии не ниже `approved`.
- Если SPEC ссылается на ADAPT через `source.adapt` — ADAPT в состоянии `approved`.

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

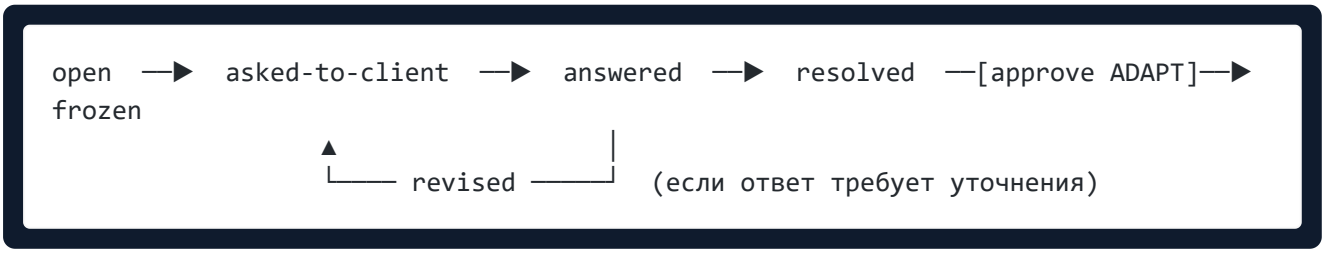


Состояния ADAPT (`draft → review → asked → answered → approved → frozen` , и терминальное `superseded` при дезавуировании) определены в главе 7 §7.4 и §7.6.4. Состояние `client-ready` изъято: вынесение вопросов клиенту — предмет ACTZ (§7.13), а не состояние ADAPT. Этот раздел нормирует gates.

Переход	Gate	Предусловие
`draft → review`	Review-transition	Прямая интерпретация охватывает все разделы Т3; первичные обратные находки зафиксированы в `open`
`review → asked`	Backward-ready	Все обратные находки переведены в `asked-to-client` ; вопросы вынесены клиенту в составе ACTZ (`status: sent`)
`asked → answered`	Client-return	Все обратные находки в `answered` ; решения зафиксированы пунктами **подписанного** ACTZ (`status: signed` , двусторонняя подпись V6)
`answered → approved`	**QG-3** (§10.4.1)	Все обратные находки в `resolved` и несут `decided-in` на пункт подписанного ACTZ; подпись архитектора 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`.
- Каждая обратная находка несёт `decided-in` на пункт ACTZ в статусе `signed` (глава 7 §7.13).
- Подпись архитектора получена и зафиксирована нативным для носителя механизмом (V3 + V6).
- Если ADAPT — delta-ADAPT: parent-ADAPT в `frozen` (глава 7 §7.6).

Постусловие:

- ADAPT переходит в `approved`.
- Запись в лог переходов с подписью архитектора.
- ADAPT, у которого хотя бы одна находка не закрыта подписанным ACTZ, **не** переводится в `approved`: интерпретация не может опираться на решение, которого клиент не принимал.

Триггер: явное одобрение архитектором.

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. Если правка затрагивает решение из подписанного ACTZ (меняет обязательство) — требуется **новый ACTZ** с двусторонней подписью; если не затрагивает — достаточно подписи архитектора.
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** (§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 (было пунктом подписанного ACTZ) — отмена **обязана** быть оформлена **новым ACTZ с двусторонней подписью**, проставляющим `superseded-by` на прежнем; подпись только архитектора допустима лишь для правки, не затрагивающей ни одного подписанного решения.
- 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 `frozen` и от `obsolete`, которым устаревают TR / SPEC / TC; 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).

Список закрывает **Quality Gates** — проверки перехода артефакта между состояниями. Обязательный релизный гейт приёмки AT (§10.4.3) в него не входит и его не расширяет: он нормирует предъявление продукта к испытаниям, а не переход артефакта. По той же причине вне списка остаются переходные события носителя (`freeze`, `deprecation`, `runner-pass` / `runner-fail`, `change-of-criteria`), фиксируемые в журнале (§10.13.1).

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 §10.4.4

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 `verified` , версия инкрементировалась)	Артефакт автоматически переведён в `approved`	V5 (version pin) для детекции
`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)
`actz-integrity` validation (§7.13)	(1) ACTZ в статусе `signed` несёт обе подписи (клиент + исполнитель), подписанты — разные лица. (2) Каждая находка в статусе `resolved` несёт `decided-in` на ACTZ в статусе `signed` ; ссылка на `draft` / `sent` / несуществующий ACTZ — fatal. (3) Каждый подписанный ACTZ отражён хотя бы в одном ADAPT: подписанное решение без отражения в интерпретации — fatal (обязательство вне требований). (4) `superseded` требует `superseded-by` .	V3 (block approve); V6 (двусторонняя подпись); V1 (подписанный ACTZ immutable)
`at-isolation` validation (§9.19)	(1) `generator` каждого AT подтверждает изоляцию: модель отлична от модели основного агента, доступ к внутреннему контуру отсутствовал — **fatal** при нарушении (иначе AT подтверждает собственную интерпретацию вместо контракта). (2) `verifies[]` ссылается только на `TZ §N` / `ACTZ-NNN §M` ; ссылка на BR / SR / SPEC / TC / ADAPT — fatal. (3) `tz-version` каждого AT совпадает с действующей редакцией итогового TZ — иначе испытания блокируются (§10.4.3).	V5 (pin к редакции итогового TZ); V6 (провенанс генератора)
`red-history` validation (§9.18.2)	(1) TC, засчитываемый как доказательство при QG-2, имеет записанный переход «красный → зелёный» либо унаследованную красную историю (задача без изменения поведения). (2) TC класса `implementation-originated` — красной истории не имеет по построению; обязателен `mutation-check.mutants-killed ≥ 1` (§6.13.3). (3) Автор TC не совпадает с автором реализации (P8, §9.18.1).	V1 (история прогонов immutable); V6 (провенанс авторства)
`adapt-applicability` validation (§7.4.1, §7.4.6)	(1) Для каждого TZ вердикт состязательного обзора выпущен как AR в статусе `issued` (V6 author +	V3 (block approve до прохождения

	timestamp). (2) Если `verdict: findings-present` — `produces-adapt[]` непуст, каждый указанный ADAPT существует в `approved` + с подписью архитектора, и каждая его находка несёт `decided-in` на подписанный ACTZ; BR/SR/SPEC производные имеют `source.adapt` . (3) Если `verdict: no-findings` — ADAPT отсутствует, BR/SR/SPEC имеют `source.tz-section` + `source.adversarial-review-ref` на эту AR. (4) Целостность AR: модель `reviewer` отлична от модели основного агента (§7.10.2); ссылка на AR в статусе `draft` либо `superseded` — fatal; висячая ссылка — fatal. (5) Запрет смешения: артефакт с опущенным `source.adapt` и без `source.adversarial-review-ref` — fatal.	validation); V6 (вердикт + signature attribution); V1 (AR immutable после `issued`)
`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 (решение было пунктом подписанного ACTZ) — отмена оформлена новым ACTZ с двусторонней подписью, проставляющим `superseded-by` на прежнем. (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 набором изменений (V4 ветвление / набор изменений, §3.3.4), регистрируемым атомарной единицей изменения (V2, §3.3.2), 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 одобрить исправление кода, которое 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`	TC	Деградация теряет history runs; используется `passing → failing → obsolete` или change-of-criteria путь
`ready`	`draft`	TC	Деградация теряет dry-run evidence (§10.9.2); ослабление TC через `[test-spec-change]` (глава 9 §9.13) — не путь возврата в `draft`

`failing`	`draft`	TC	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` / `ACTZ` / `AR` / `TC` / `AT`	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` / `at-release` (§10.4.3) / `deprecation` / `freeze` / `runner-pass` / `runner-fail` / `change-of-criteria`	Обязательно
`actor`	Идентификатор инициатора (V6); для двусторонней подписи ACTZ — 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); подпись архитектора (§7.5); ACTZ и двусторонняя подпись (§7.13); 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 (§10.4.4); хранит retention policy событий (§10.13.3)

---

11. Модель зрелости

> ***Часть RENAR Standard v1.0*** · *← 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; Т3 зафиксировано
`RENAR-3`	Учитываемый (Tracked)	Полная схема frontmatter; статусы жизненного цикла используются; delta-Т3 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); каждое ТЗ прошло состязательный обзор, его исход выпущен как AR, и ADAPT создан при вердикте «findings present» (§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); TC расширения `tc-type`; pos/neg парность TC.

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[].requirement-version` (§9.3, 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 проверяться автоматически).

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 (§9.3); 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 подписью архитектора и ACTZ с двусторонней 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` — требуется, но без обеспечения соблюдения носителем (проверка процедурная); ✓ — нормативно обязательно.

Признак	RENAR-1	RENAR-2	RENAR-3	RENAR-4	RENAR-5
Носитель с V1–V6	✓	✓	✓	✓	✓
Состязательный обзор T3 с выпущенной AR; ADAPT — по вердикту	✓	✓	✓	✓	✓
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 зафиксированного вердикта состязательного обзора (AR) для каждого Т3 — **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 TC 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 TC для каждого нормативного утверждения.
2. Реализация TC 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 TC).
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	Реактивный ADAPT — обязательное положение независимо от уровня (§11.4.1): состязательный обзор на каждое T3, ADAPT по вердикту; подпись архитектора на ADAPT (QG-3) — declared на RENAR-4+; двусторонняя подпись — на ACTZ (§7.13)
08 Specifications	Закрытый список 11 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*** · *← 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 T3, ни в ADAPT, — пока он не всплывёт на приёмке спором с клиентом. «Процесс в целом» здоров, а работа с требованиями — нет. Поэтому RENAR добавляет **одиннадцать 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-T3 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 одиннадцати 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 T3 в носителе до состояния «все BR/SR (parent цепочка из этого T3) → `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 T3; 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 (T3/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).

12.3.11 Implementation-originated Rate

Доля требований класса `source: implementation-originated` (§6.13) среди всех требований, добавленных за период. **Формула:** `IOR = count(SR где source=implementation-originated) / count(all SR added) × 100%`. **Источник:** frontmatter `source` + журнал аудита.

Зачем метрика обязательна. Класс `implementation-originated` — узкая легализация внутренних технических деталей, возникших при реализации. Он полезен, пока редок: его рост означает, что **процесс требований протекает** — функциональность рождается в коде, а не в требованиях, и легализуется задним числом. Норма §6.13 требует, чтобы этот дрейф был **виден**, а не размазан по проекту незаметно.

Цели: RENAR-3 ≤ 15%; RENAR-4 ≤ 5%; RENAR-5 ≤ 2%. Превышение — не нарушение соответствия само по себе, но обязано разбираться на ревизии (§12.5.3): либо требования пишутся недостаточно полно, либо граница «наблюдаемое клиентом» трактуется слишком широко.

12.4 Сводная таблица целей по уровням

Закрытый список 11 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

Implementation-originated Rate	≤ 15%	≤ 5%	≤ 2%	frontmatter `source` + журнал аудита
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*** · *← Оглавление*

> ***Плотная глава:*** *начните с комплекта самооценки reference/08; соответствие MVR ↔ §13.3 — 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`; T3 зафиксировано	Соответствие декларируем (§13.2.2)
`RENAR-3`	Tracked (ведётся жизненный цикл и учёт связей): полная `frontmatter`-схема; статусы жизненного цикла используются; delta-T3 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, 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

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 невозможны подпись архитектора на ADAPT, двусторонняя подпись ACTZ и 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 T3 обязано пройти состязательный обзор (§7.10.2); его исход обязан быть выпущен как AR — запись состязательного обзора в статусе `issued` (V6 author + timestamp, §7.4.6). 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` с подписью Архитектора (§7.5). Каждая находка несёт `decided-in` на пункт **подписанного** ACTZ (§7.13). Lifecycle §7.4.5. AR несёт непустой `produces-adapt[]`.	`source.adapt` mandatory; `source.tz-section` mandatory
«no findings, no clarifications» — конвертация T3 → RENAR однозначна	ADAPT не создаётся	`source.tz-section` mandatory; `source.adversarial-review-ref` mandatory — ссылка на AR с вердиктом `no-findings`

Делегирование на состязательного рецензента — единственный допустимый способ задекларировать «ADAPT не нужен». Создание BR / SR / SPEC из T3 без зафиксированного вердикта — нарушение стандарта; hooks (§10.11.1 `adapt-applicability validation`) обязаны блокировать такие артефакты.

При наличии delta-T3 — то же правило (§7.6): delta-ADAPT создаётся по факту находок состязательного обзора delta-T3.

Множественность и стадийная независимость. Триггер ADAPT стадийно-независим: вердикт выносится не только при импорте T3, но и на стадиях деривации (BR → SR → SPEC → TC, §7.4.1.1). На одно T3 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 (решение было пунктом подписанного ACTZ) — быть оформленным **новым ACTZ с двусторонней подписью**, проставляющим `superseded-by` на прежнем; (3) перенаправить или пере-вывести все производные так, чтобы не осталось висячих

`source.adapt` на `superseded` (§6.10.3). Отдельный QG не требуется — используется QG-3 (§10.8.5).

Негативные сценарии (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».
- `source.adversarial-review-ref` указывает на несуществующую AR, либо на AR в статусе `draft` (вердикт не выпущен), либо на AR в статусе `superseded` (вердикт замещён) — невалидное свидетельство (§7.4.6).
- AR выпущена с моделью рецензента, совпадающей с моделью основного агента, — обзор не является состязательным (§7.10.2).
- AR с `verdict: findings-present` и пустым `produces-adapt[]` — вердикт заявляет находки, но ни один ADAPT не порождён.
- Находка в статусе `resolved` без `decided-in` на подписанный ACTZ — интерпретация опирается на решение, которого клиент не принимал (§7.4.5).
- Подписанный ACTZ, ни одно решение которого не отражено в ADAPT, — обязательство существует вне требований.
- Подпись клиента на ADAPT объявлена обязательной — избыточное требование; допустимо только как `declared-stricter` (§1.7.2), но не как базовое соответствие.
- **АТ сгенерирован агентом, имевшим доступ к внутреннему контуру** (ADAPT / BR / SR / SPEC / TC / код) — уровень приёмки схлопывается в уровень верификации, и ошибка интерпретации получает ложное подтверждение соответствия (§9.19.2).
- Продукт предъявлен к испытаниям при АТ, выведенных из **устаревшей** редакции итогового ТЗ (§9.19.4).
- TC засчитан как доказательство **без красной истории** и без убитого мутанта (§9.18.2).
- Требование класса `implementation-originated` описывает **наблюдаемое клиентом** поведение (§6.13.2) — обязательство перед клиентом не может возникнуть из кода.
- 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 нового подписанного ACTZ — односторонняя отмена согласованного с клиентом, нарушение §7.6.4.

13.3.4 Закрытый список 11 типов SPEC

Тип SPEC обязан принадлежать закрытому списку одиннадцати типов (§8.3): `SPEC-ARCH`, `SPEC-API`, `SPEC-DATA`, `SPEC-INT`, `SPEC-PROC`, `SPEC-UI`, `SPEC-AI`, `SPEC-SEC`, `SPEC-OPS`, `SPEC-TEST`, `SPEC-DOC`.

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 подписью архитектора на 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:
```

```yaml
  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 (§10.4.4)
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",
                       "SPEC-TEST", "SPEC-DOC"]
# Все 11 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 обязательных положений (§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 `source.adapt` и без `source.adversarial-review-ref` — молчаливый пропуск состязательного обзора; 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 нормативный триггер)
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- или major-версии стандарта → 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 (реактивный ADAPT: состязательный обзор на каждое ТЗ, ADAPT — по вердикту); §7.4.4 закрытый список backward findings — §13.3.7
08 Specifications	Обязательное положение §13.3.4 (закрытый список 11 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, §10.4.4 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*** · *← 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** (§1.6.1) 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. §1.6.1
QG-0 / QG-1 / QG-2 как концепция	RENAR конкретизирует машины состояний в §10
10 общих метрик процесса	RENAR добавляет 11 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 издателем* — 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 нефункционального SR (§6.6.2) обязан содержать `quality-characteristic` из перечня 25010; для функционального SR поле опускается.

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 следствия 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