RENAR

Справочные приложения (глоссарий, схемы, реестры)

RENAR · Справочник · Версия 1.0-draft | 31.05.2026

Авторы: Вадим Соглаев, Андрей Юмашев

Глоссарий RENAR

> ***Назначение:*** *единый источник канонических терминов RENAR с примерами и mapping на отраслевые стандарты. При расхождении формулировок* ***нормативно*** *побеждает* `standard/04-terms.md` *; этот глоссарий — informative lookup для читателя и assessor-а.*

1. Цепочка авторитетности

В случае разногласий по термину применяется порядок:

1. `standard/04-terms.md` — нормативный канон: определения главы стандарта побеждают при любом расхождении.
2. **Этот документ** — информационные разъяснения и сопоставление с отраслевыми стандартами; не переопределяет `standard/04` .
3. **ISO/IEC/IEEE 29148** — международный стандарт инженерии требований.
4. **BABOK Guide v3** — свод знаний по бизнес-анализу (*Business Analysis Body of Knowledge*).
5. **Изменение через предложение поправки в полный стандарт RENAR** — если все источники молчат.

Закрытые списки: главный индекс шестнадцати закрытых списков — `standard/01 §1.7.5` .

Не используются как источник терминов: тикеты в багтрекере, чаты команды (slang), устаревшие презентации, маркетинговые материалы.

2. Канонические термины

2.1 Уровни требований

Каноника v1.0 — закрытый список (см. `standard/04-terms.md §4.3`):

RENAR (канонический)	Полное название	RU (для UI)	Описание
BR	Business Requirement	Бизнес-требование	Бизнес-цель уровня заказчика: что организация хочет получить.
SR	System Requirement	Системное требование	Инженерное требование к ПО; поддаётся проверке. Один `SR` — одна проверяемая единица.
TR	Task Requirement	Требование к задаче	Требование уровня реализации, выводимое из `SR` ; единица планирования работ.

TC	Test Case	Контрольный пример (TC)	Проверяемый артефакт, подтверждающий `SR` / `BR`. Описывает поведение, а не реализацию.

Правило идентификации: в `frontmatter` `id` — канонический идентификатор RENAR (`BR-01`, `SR-05`). При экспорте в другой носитель применяется целевое сопоставление.

Уточнения для `SR` (по полям frontmatter, не отдельные типы артефактов):

- **Module SR** — `SR` с `level: module` (`standard/06 §6.7`). Раньше отдельный label `TM` (§2.1.1).
- **Integration SR** — `SR` с `constrained-by: [SPEC-INT-N]`. Раньше отдельный label `INT-SR` (§2.1.1).
- **Contract TC** — `TC` с `tc-type: contract`. Раньше отдельный label `INT-TC` (§2.1.1).

Семейство `SPEC` (UX / ИИ / архитектура / интеграционные спецификации) — см. §14.5 Семейство `SPEC`.

2.1.1 Legacy labels (deprecated)

При миграции с материалов до v1.0 draft встречаются устаревшие метки. Их замены в канонике v1.0 (`standard/04-terms.md §4.14.1`):

Устаревший label	Замена в канонике v1.0	Пояснение
`TM` (Module/Submodule SR)	`SR` с `level: module`	Уточнение `SR`, не отдельный тип артефакта
`UIC` (UI Concept)	`SPEC-UI` (`standard/08 §8.5.6`)	Часть семейства `SPEC` (§14.5)
`AIC` (AI Concept)	`SPEC-AI` (`standard/08 §8.5.7`)	Часть семейства `SPEC`
`INT-SR` (Integration SR)	`SR` с `constrained-by: [SPEC-INT-N]`	Подкласс `SR`
`INT-TC` (Integration TC)	`TC` с `tc-type: contract`	Подкласс `TC`
`TS` (Technical Specification)	`SPEC-ARCH` или `SPEC-OPS` в зависимости от содержания	Часть семейства `SPEC`

Миграция существующих артефактов: встроенная в носитель привязка к набору изменений должна автоматически распознавать устаревшие метки и предлагать канонические замены. Каноническая таблица сопоставления legacy → v1.0 — `standard/04-terms.md §4.14.1`.

2.2 ADAPT artifact family

Термин	Описание
ADAPT	Адаптивный документ, формулирующий проектное ТЗ (*Adaptive Document for Articulating Project's TZ*). Двусторонняя инженерная адаптация ТЗ: прямое направление (интерпретация) и обратное (вопросы). Утверждается двойной подписью.

delta-ADAPT	`ADAPT` для delta-ТЗ. Цепочка: ADAPT-001 → ADAPT-001-delta-1 → ADAPT-001-delta-2; применяются по порядку.
errata-ADAPT	Правка утверждённого `ADAPT` (наша ошибка интерпретации). Не меняет замороженный документ — добавляется отдельный артефакт.
Forward (в ADAPT)	Инженерная интерпретация каждого раздела ТЗ: цитата → интерпретация → достроенные сценарии → охват.
Backward (в ADAPT)	Список проблем и вопросов к клиенту. Жизненный цикл: `open` → `asked-to-client` → `answered` → `resolved` → `frozen`.
Term mapping (в ADAPT)	Таблица «термин клиента → инженерное понимание».

2.3 Backward categories (закрытый список)

ADAPT backward записи относятся к одной из 7 категорий. Список закрыт; новые добавляются только через изменение полного RENAR Standard:

Категория	Описание
`contradiction`	Противоречие внутри ТЗ.
`gap`	Гэп — про это ТЗ молчит.
`hidden-assumption`	Скрытое предположение инженера, требующее подтверждения.
`feasibility`	Технически нереализуемое или дорогое требование.
`regulatory`	Затрагивает законодательство / compliance.
`terminology`	Неясный или конфликтующий термин клиента.
`scope`	Уточнение границ scope (входит / не входит).

2.4 Контрольные точки качества (`Quality Gates` , закрытый список, каноника v1.0)

Закрытый список контрольных точек RENAR (по `standard/10-lifecycle-qg.md §10.3–10.4`):

Gate	Применяется к	Условие пропуска
QG-0 Контрольная точка утверждения (`Approval Gate`)	Перевод `BR` / `SR` / `SPEC` : `draft` → `approved`	Цель, критерии приёмки, не менее одного негативного сценария и охват; для `SR` — родительский `BR` в `approved` +; для `SR` с `constrained-by` — `SPEC` в `approved` +.
QG-1 Контрольная точка реализации (`Implementation Gate`)	Перевод `TC` : `draft` → `ready` (**только** для `TC`)	У `TC` поле `verifies` ссылается на артефакт в `approved` +; `requirement-version` совпадает; охват реализации и закрепление версии зафиксированы.

QG-2 Контрольная точка проверки (`Verification Gate`)	Перевод `SR` / `BR` : `approved` → `verified`	Все `TC` из `verified-by` зелёные; хотя бы один `TC` с `negative: true` ; у `last-run` совпадает `requirement-version` с текущей `version` ; выборочная проверка пройдена.
QG-3 Контрольная точка архитектуры (`Architecture Gate` , *опционально*)	Утверждение `SPEC-ARCH` / двойная подпись	Зафиксирован артефакт в духе ADR в носителе; двойная подпись (клиент + архитектор). Не обязательна для декларируемого соответствия RENAR.
QG-4 Контрольная точка приёмки (`Acceptance Gate` , *опционально*)	Перевод `BR` : `verified` → `accepted`	Все `BR` покрыты проверенными `SR` ; приёмочные `TC` (`tc-type: acceptance`) зелёные; подпись клиента. Не обязательна для декларируемого соответствия RENAR.

Соответствие стандарту RENAR (соответствие): `QG-0` / `QG-1` / `QG-2` обязательны для декларируемого соответствия (`standard/13 §13.3`). `QG-3` / `QG-4` — необязательные расширения.

Список закрыт; новые номера контрольных точек добавляются только через формальную процедуру изменения полного стандарта RENAR.

2.4.1 Устаревшие имена QG (deprecated)

До v1.0 в RENAR использовались иные имена контрольных точек. Сопоставление по `standard/04-terms.md §4.14.1` :

Устаревшее (до v1.0)	Замена в каноникe v1.0	Пояснение
`QG-0 Context Gate`	`QG-0 Approval Gate`	Только переименование; смысл сохранён
`QG-1 Requirements Gate`	`QG-1 Implementation Gate`	**Сдвиг смысла:** ранее утверждение `BR` / `SR` , теперь только перевод `TC` : `draft` → `ready`
`QG-2 Implementation Gate`	`QG-1 Implementation Gate`	Перенумерация и слияние в одну `QG-1`
`QG-3 Verification Gate`	`QG-2 Verification Gate`	Перенумерация
`QG-4 Acceptance Gate`	`QG-4 Acceptance Gate`	Имя то же; теперь **опционально** для декларируемого соответствия RENAR

Миграция существующих артефактов: средства автоматизации преобразуют ссылки по таблице выше. Каноническая таблица сопоставления legacy QG → v1.0 — `standard/04-terms.md §4.14.1` .

2.4.2 Локальные контрольные точки ADAPT

Жизненный цикл `ADAPT` (`standard/07-adapt.md`) использует локальные контрольные точки `ADAPT` параллельно каноническим `QG-N` . Это не отдельная нумерация `QG` , а локальные события жизненного цикла артефакта `ADAPT` :

Контрольная точка	Применение	Условие прохода
QG-ADAPT-draft	Создание `ADAPT`	Раздел Forward охватывает все разделы T3.
QG-ADAPT-review	Перевод в `review`	Все записи backward — `open` или `asked-to-client`; нет состояния `draft`.
QG-ADAPT-client-ready	Передача клиенту	Все backward — `asked-to-client`; пакет вопросов собран.
QG-ADAPT-answered	После ответа клиента	Все backward — `answered`.
QG-ADAPT-approve	Утверждение `ADAPT`	Все backward — `resolved`; двойная подпись (клиент + архитектор).
QG-ADAPT-frozen	После утверждения	Неизменяемость; разрешена генерация `BR` / `SR` / `SPEC`.

Локальные контрольные точки `ADAPT` **не входят** в набор `QG-0` / `QG-1` / `QG-2` для декларируемого соответствия RENAR. Реализация может выразить `QG-ADAPT-approve` как локальный псевдоним для `QG-3` (`Architecture Gate` , если применимо) либо хранить отдельно — на усмотрение носителя.

2.5 Семейство SPEC (закрытый список)

Тип	Назначение	Источник
SPEC-ARCH	Архитектура системы / подсистемы: контексты, контейнеры, компоненты, представление развёртывания, атрибуты качества	Раздел Forward `ADAPT`
SPEC-API	Контракты API (REST / GraphQL / gRPC / асинхронные события); версионирование, модель ошибок, ограничения частоты	Раздел Forward `ADAPT`
SPEC-DATA	Модель данных: схема, ER-диаграмма, индексы, миграции, хранение, классификация персональных данных	Раздел Forward `ADAPT`
SPEC-INT	Интеграция: взаимодействие между подсистемами и внешними системами; протоколы, контракты, SLA	Раздел Forward `ADAPT`
SPEC-PROC	Процесс / рабочий поток: бизнес-процессы, машины состояний, паттерны saga, оркестровка и хореография	Раздел Forward `ADAPT`
SPEC-UI	UI / UX: экраны, навигация, пользовательские сценарии, доступность, локализация, эталонные изображения	Раздел Forward `ADAPT`
SPEC-AI	ИИ / ML: карточки моделей, RAG, инженерия промптов, стратегия оценки, бюджет стоимости	Раздел Forward `ADAPT`
SPEC-SEC	Безопасность: аутентификация / авторизация, модель угроз, управление секретами, классификация данных	Раздел Forward `ADAPT`, регуляторные замечания backward

SPEC-OPS	Эксплуатация: развёртывание, наблюдаемость, SLO / SLA, runbook, аварийное восстановление	Раздел Forward `ADAPT`

Список закрыт — ровно девять типов (каноника по `standard/08 §8.3`); новые типы SPEC добавляются только через изменение полного стандарта RENAR.

2.6 Статусы жизненного цикла

Статус	Применим к	Значение
`draft`	все артефакты	В работе, не для использования другими
`review`	все	На ревью, возможны изменения
`approved`	ADAPT, SR, SPEC	Утверждён; неизменяем после двойной подписи (для `ADAPT`) или одной (для `SR` / `SPEC`)
`verified`	SR	Подтверждён успешными `TC` и выборочной проверкой
`frozen`	ADAPT	Утверждён и неизменяем; используется как источник для генерации `BR` / `SR` / `SPEC`
`deprecated`	все	Устарел, не используется в новых артефактах; замена (если есть) указывается полем `replaced-by`
`obsolete`	research/legacy	Полностью изъят из обращения

2.7 Возможности носителя (V1–V6)

RENAR не привязан к конкретному носителю артефактов. Артефакт может находиться в любом носителе, который удовлетворяет следующим возможностям:

Нормативное определение — `standard/03 §3.3`:

Возможность	Описание
V1 — неизменяемая история	Любое прошлое состояние артефакта можно адресно восстановить без потерь (цепочка ревизий для аудита сохраняется).
V2 — атомарная единица изменения	Изменение артефакта (или согласованной группы) фиксируется одной транзакцией: целиком успех или целиком откат; промежуточные состояния извне не видны.
V3 — сравнение и ревью (diff)	Предлагаемое изменение можно представить как разницу к базовой версии и принять или отклонить до попадания в утверждённое состояние (основа утверждения и контрольных точек `QG`).
V4 — ветвление и набор изменений	Незавершённая работа отделена от утверждённого источника истины; несколько независимых изменений ведутся параллельно без влияния на источник истины.
V5 — межсценарная фиксация версии	Можно зафиксировать конкретную версию артефакта из другого носителя как разрешимый идентификатор (основа поля `verifies[].requirement-version`).

V6 — автор и метка времени	Для каждой атомарной правки зарегистрированы однозначный автор и метка времени (основа подписи `ADAPT` и блока `ai-provenance`).

Примеры сред: распределённая VCS (`git` с ревью запросов на слияние), документоориентированное хранилище с цепочкой ревизий и подписями, любая СУБД с историей изменений и подписями. RENAR не требует `git` — только возможности **V1–V6**.

2.8 Происхождение ИИ (`ai-provenance`, канонические поля)

Во `frontmatter` любого артефакта, сгенерированного ИИ:

Поле	Тип	Описание
`ai-provenance.generated-by`	string	Модель в формате `<vendor>-<model>-<version>@<date>`. Пример: `anthropic-claude-opus-4-7@2026-05-15`.
`ai-provenance.prompt-template`	string	Путь к шаблону промпта и версия. Пример: `prompts/adapt-from-tz.md@v2.1`.
`ai-provenance.context-tokens`	integer	Число токенов во входном контексте.
`ai-provenance.output-tokens`	integer	Число токенов в выводе модели.
`ai-provenance.generation-time-ms`	integer	Время генерации в миллисекундах.
`ai-provenance.human-edits`	boolean	Значение «истина», если текст после генерации правил человек. **Для утверждённых артефактов обязательно `true`.**

2.9 Типы контрольных примеров

Закрытый список — шесть типов (каноника по `standard/09 §9.5`):

Тип TC (`tc-type` поле)	ISTQB соответствие	Применение
`acceptance`	Acceptance Testing	Проверка BR (приёмочный).
`ux`	расширение по удобству	Проверка `SPEC-UI` через модель-судью VLM / визуальное сравнение с эталоном (`baseline`).
`system`	System Testing	Проверка SR, SPEC-PROC, SPEC-ARCH.
`contract`	Component Integration Testing	Проверка SPEC-API / SPEC-INT / SPEC-DATA через contract testing (Pact и аналоги).
`eval`	специфично для ИИ	Проверка `SPEC-AI` через оценивающую LLM и метрики (BLEU, точность, доля галлюцинаций).

`security`	расширение по безопасности	Проверка `SPEC-SEC` : инварианты безопасности (STRIDE); нормативно только негативные сценарии (`standard/09 §9.6.4`).

2.10 Связи (frontmatter поля)

Поле	Назначение
`parent`	Родитель в иерархии (BR → SR → TR); один источник.
`children`	Дочерние артефакты (выводятся автоматически).
`source.adapt`	`ADAPT` , из которого выведен артефакт (каноника для `BR` / `SR` / `SPEC`).
`source.adapt-section`	Раздел `ADAPT` (Forward §N).
`source.tz-section`	Раздел ТЗ (справочно, для трассируемости).
`verifies` (в `TC`)	`SR` / `BR` , которые покрывает `TC` , с указанием `requirement-version` .
`verified-by` (в `SR`)	`TC` , подтверждающие `SR` (выводятся автоматически).
`derived-from`	Шаблон и версия (если артефакт создан из шаблона).
`replaces` / `replaced-by`	Замена при статусе `deprecated` .
`supersedes` (в новом)	Какое требование заменяется.
`linked-tasks`	Задачи, реализующие `SR` (через среду выполнения, не через файлы).

2.11 Соглашение об именах файлов (по умолчанию)

Тип	Шаблон	Пример
ADAPT	`adapt/ADAPT-NNN[-delta-N].md`	`adapt/ADAPT-001-main.md` , `adapt/ADAPT-001-delta-1.md`
BR	`br/BR-NN-<slug>.md`	`br/BR-01-notification-capture.md`
SR	`sr/SR-NN-<slug>.md`	`sr/SR-05-notification-feed.md`
SR подсистемы	`sr/<MODULE>-SR-NN.N-<slug>.md`	`sr/WMS-SR-01.2-pick.md`
SPEC-UI	`specs/ui/SPEC-UI-NN-<slug>.md`	`specs/ui/SPEC-UI-02-notification-feed.md`
SPEC-AI	`specs/ai/SPEC-AI-NN-<slug>.md`	`specs/ai/SPEC-AI-01-rag-strategy.md`
SPEC-INT	`specs/int/SPEC-INT-NN-<slug>.md`	`specs/int/SPEC-INT-01-auth-billing.md`
SPEC (generic)	`specs/<type>/SPEC-<KIND>-NN-<slug>.md`	`specs/api/SPEC-API-03-orders.md`
TC	`tests/TC-NN-<slug>.md`	`tests/TC-01-login-success.md`

ТЗ	`tz/TZ-YYYY-NNN.md`	`tz/TZ-2026-001.md`
Дельта-ТЗ	`tz/TZ-YYYY-NNN-delta-N.md`	`tz/TZ-2026-001-delta-1.md`
UX-эталон	`specs/ui/baselines/SPEC-UI-NN-<scenario>.png`	`specs/ui/baselines/SPEC-UI-02-feed-default.png`
Eval dataset	`specs/ai/eval-datasets/SPEC-AI-NN-<slug>.jsonl`	`specs/ai/eval-datasets/SPEC-AI-01-typical-queries.jsonl`

Соглашение по умолчанию; хранилища, встроенные в конкретную среду, могут использовать иную раскладку при условии устойчивых идентификаторов (возможность **V3**).

2.12 Маркеры записи об изменении (справочно)

В носителе, где атомарные изменения сопровождаются метаданными (текст коммита в `git`, описание записи об изменении в документоориентированном хранилище), допустимы маркеры:

Маркер	Назначение
`[delta:TZ-YYYY-NNN]`	Изменение по delta-ТЗ.
`[test-spec-change]`	Изменение критериев типа «пройдено / не пройдено» у `ТС` (отдельное утверждение).
`[baseline-update]`	Обновление эталона UX или набора для оценки (отдельное утверждение).
`[coverage]`	Автоматическая регенерация сводок по покрытию, плану контрольных примеров и требованиям (бот).
`[reconciliation]`	Изменение от агента согласования (reconciliation).
`[multi-model-disagreement]`	Артефакт с расхождением ответов моделей ИИ — требует ручного ревью.
`[AI]`	Префикс для изменений, сгенерированных ИИ.

Механизм, встроенный в носитель, может выразить эти маркеры полями записи об изменении, метками или тегами — детали не нормируются.

3. Сопоставление со стандартами

3.1 Уровни требований

Метки каноники v1.0 RENAR и внешние стандарты:

RENAR	ISO/IEC 29148	BABOK	SAFe	Document store (example enum)	SENAR (RU)
BR	Business Requirement	Business Need	Portfolio Epic / Strategic Theme	`BT`	БТ

SR	System / Software Requirement	Solution Requirement (Functional)	Feature	`ST`	CT
SR (`level: module`)	(область подкомпонента, расширение)	(область подкомпонента)	Story (иногда)	`TM` (legacy)	CT модул
SR (`constrained-by: SPEC-INT-N`)	Требование к интерфейсу	Интерфейсное решение	Межфичевая интеграция	(связано)	INT-CT (legacy
TR	(нет прямого класса; детализация требования к системе / элементу системы)	Детализированное требование к решению	Story	`TK`	Т3
TC	Контрольный пример (Test Case)	Верификация	Приёмочный тест истории	`test_case`	ТК
TC (`tc-type: contract`)	Тест интерфейса	Component Integration Testing	Контрактный тест	(связано)	INT-TK (legacy
SPEC-UI	(между BR/SR — проектная спецификация)	Требование стейкхолдера (фрагмент UX)	(уровень проектирования)	(расширение)	UIC (legacy
SPEC-AI	(расширение REQ)	(н/д)	Enabler	(расширение)	AIC (legacy
SPEC-ARCH / SPEC-OPS	Описание проектирования	Компонент решения	Enabler tech spec	(расширение)	TC (legacy

Примечание: колонки «document store (пример перечисления)» и «SENAR (RU)» содержат исторические метки (`TM` , `UIC` , `AIC` , `INT-CT` и т.д.) для трассируемости с системами до v1.0. Колонка канониki RENAR — метки v1.0 согласно §2.1.1. При экспорте в document store или в носитель SENAR применяют целевое сопоставление.

3.2 Контрольные точки качества

Сопоставление канонических `QG-N` v1.0 RENAR с внешними моделями. Устаревшие имена `QG` (§14.4.1) в колонке SENAR сохранены для исторической трассируемости.

RENAR (v1.0)	SENAR (сопоставление с legacy)	Document store (пример)	Активность CMMI
QG-0 Контрольная точка утверждения	QG-0 (контекст)	`VK-1` (старт)	Проверка требований до обязательств

QG-1 Контрольная точка реализации	QG-2 (реализация в legacy)	`VK-1`	Базовая линия реализации (готовность `TC`)
QG-2 Контрольная точка проверки	QG-3 (верификация в legacy)	`VK-2`	Верификация
QG-3 Контрольная точка архитектуры *(опционально)*	(н/д)	`VK-3` (частично)	Утверждение архитектурного решения
QG-4 Контрольная точка приёмки *(опционально)*	QG-4 (Приёмка)	`VK-4`	Приёмка заказчиком

Примечание: до v1.0 `QG-1 Requirements Gate` фактически разделён между каноническим `QG-0 Approval Gate` (утверждение `BR` / `SR` / `SPEC`) и `QG-1 Implementation Gate` (готовность `TC`). См. §14.4.1 для полного сопоставления.

3.3 Статусы жизненного цикла

RENAR	Document store (example)	ISO/IEC 29148	CMMI
`draft`	`draft`	proposed	identified
`approved`	`approved`	agreed-to / baselined	committed
`verified`	`verified`	verified	validated
`deprecated`	`obsolete`	retired	obsolete

3.4 Артефакты процесса

RENAR	BABOK	SAFe	SENAR
ADAPT (Forward + Backward)	Requirements Analysis Document	Solution Intent (fixed + variable)	(n/a — RENAR-extension)
Work Order / T3	Stakeholder commitment artifact	Customer order	(контекст)
delta-TZ	Change Request	(n/a — handled via Solution Intent updates)	(контекст)
Impact Analysis	Impact Analysis (BABOK §8)	(derived)	(контекст)
Выборочная проверка	Random sampling QA	(n/a)	Rule 9.5
Состязательный обзор	Independent verification	(n/a — REQ-extension)	(via ADR metric)
Reconciliation	Continuous improvement audit	Inspect & Adapt	Quality Sweep

3.5 Переводы в пользовательском интерфейсе

Поля `frontmatter`, идентификаторы и имена файлов — всегда канонические (латиница). В интерфейсе допустимы русские подписи:

Канонический	UI (RU)
Business Requirement	Бизнес-требование
System Requirement	Системное требование
Test Case	Контрольный пример (`TC`)
Quality Gate	Контрольная точка качества
Acceptance	Приёмка
Verified	Проверено
Approved	Утверждено
Deprecated	Устарело
Frozen	Замороженный
Backward finding	Замечание к ТЗ
Forward interpretation	Инженерная интерпретация

Перевод в интерфейсе не заменяет канонические идентификаторы в носителе.

4. Запрещённые / устаревшие термины

RENAR не использует следующие термины (даже если они встречаются в SENAR / отраслевой литературе):

Термин	Что использовать вместо	Почему
User Story как требование	SR	Story — единица планирования, не требование. Story может реализовывать SR, но сама требованием не является.
Use Case (формально)	SPEC-UI + SR	Use case смешивает UX и поведение. RENAR разделяет SPEC-UI (UX-эталон) и SR (поведение).
Spec (без квалификатора)	SR / BR / SPEC-API / SPEC-DATA / ...	«Spec» неоднозначно. Используем точные термины.
Бизнес-логика как требование	SR	«Бизнес-логика» — термин кода, не требований.
Функциональность	`SR` / `TR`	Слишком широко; не поддаётся однозначной проверке.
Фича (mixed-lang)	Feature (SAFe context) или SR (канонический термин RENAR)	Mixed Russian/English; неоднозначно.

Хотелка	(никогда)	Договорный документ так не пишется.
Эпик как требование	BR (бизнес-уровень) или Portfolio Epic (SAFe)	Epic — единица планирования, не требование.
«Тестировать руками»	выборочная проверка (Rule 5 Core) или ручной TC с `type: acceptance`	Расплывчатое; нет проверяемого свидетельства.
«Доделать» (как статус)	`draft` / `review`	Не из закрытого списка жизненного цикла.
TODO во `frontmatter`	запись backward в `ADAPT` (если речь о требовании) или задача в трекере (если о реализации)	Открытые вопросы живут в нужном артефакте, не в комментарии к полю.

При таких находках в локальных артефактах проекта включайте предупреждение на стороне носителя (`pre-commit` в `git` , правило валидации в document store).

---

5. Версионирование глоссария

Глоссарий — самостоятельный документ со своей версией. Изменение канонического термина — скачок major-версии (1.0 → 2.0) и сценарий миграции для всех проектных артефактов.

Текущая версия: 1.0-reconciled (согласование фазы 1.5 с `standard/04-terms.md §4.14.1`).

5.1 Открытые вопросы — закрыты (фаза 1.5, 2026-05-16)

Четыре открытых вопроса прежней редакции черновика закрыты согласованием с `standard/04-terms.md §4.14.1` :

#	Был открытым вопросом	Итог	Источник
1	Канонический язык: латиница (`BR` / `SR`) или русский (БТ/СТ)?	**Канон — латиница**; русский только в проекции UI (§4.5)	`standard/04 §4.13.3` + `reference/06-ru-style-guide.md §1.3`
2	`TM` как отдельная метка или уточнение `SR` ?	**`SR c level: module`** — уточнение, не отдельный тип артефакта	`standard/04 §4.14.1` (устарело `TM`) + §2.1.1
3	AIC ← AAC / AIA / AIC?	**`SPEC-AI`** (каноника v1.0); AIC — устаревшая метка	`standard/04 §4.14.1` + §2.1.1
4	`INT-TC` отдельный тип или соглашение об именах?	**`TC c tc-type: contract`** — уточнение, не отдельный тип	`standard/04 §4.14.1` (`INT-TC` deprecated) + §2.1.1

5.2 История согласований

Дата	Версия	Изменение

2026-05-16	1.0-reconciled	Согласование фазы 1.5 с `standard/04-terms.md §4.14.1`. Устаревшие метки перенесены из таблицы `§14.1` в `§14.1.1`; имена `QG` приведены к канонике v1.0 в §14.4; устаревшие имена → `§14.1`. Обновлены таблицы сопоставления §4.1 и §4.2. Четыре открытых вопроса закрыты (§2.1). Задача: `ru-reconcile-glossary-vs-standard`.
(ранние черновики)	1.0-draft	Наполнение черновика в фазе 7; были открытые вопросы и расхождения с `standard/04 §4.14.1`. История коммита зафиксирована; заменено выпуском 1.0-reconciled.

5.3 Перекрёстные ссылки

- **Стиль редакции v1.0** (`reference/06-ru-style-guide.md`) — правила русской редакции нормативного текста; §1.9 задаёт канонический перечень терминов параллельно этому глоссарию. При расхождении приоритет формулировок для редакционных проходов — у руководства по стилю.
- **Канонические определения** (`standard/04-terms.md`); §4.14.1 — сопоставление устаревшие → каноника.

Глоссарий RENAR 1.0-reconciled — часть `reference/`. См. также 02-schemas.md, 03-ai-risk-register.md, 06-ru-style-guide.md.

Схемы (формальные)

> ***Назначение:*** *машино-читаемые YAML frontmatter схемы для всех типов артефактов RENAR. Используются нативными для носителя валидаторами для проверки соответствия.*
> ***Нормативные определения структуры*** *— в* `standard/06`, `standard/07`, `standard/08`, `standard/09`. *Валидация примеров:* `node scripts/validate-schema-examples.js`. *Этот документ — справочник (informative lookup).*

1. Общий frontmatter (все артефакты требований и SPEC)

Поля, общие для BR/SR/TR/SPEC-* (канонические v1.0). Legacy типы `UIC` / `AIC` / `INT-SR` / `TS` deprecated в v1.0 (standard/04 §4.14.1).

```yaml
# Identity
id: "<TYPE>-NN[.N]"
title: "<short, descriptive>"
type: BR | SR | TR | SPEC-ARCH | SPEC-API | SPEC-DATA | SPEC-INT | SPEC-PROC |
SPEC-UI | SPEC-AI | SPEC-SEC | SPEC-OPS
slug: "<kebab-case>"

# Scope
level: system | subsystem | module
scope: { system: "<system-id>", subsystem: "<subsystem-id>" }   # subsystem=null
если level=system

# Жизненный цикл (verified — BR/SR; frozen — ADAPT §7; ready/passing/failing — TC
§8; см. standard/04 §4.6)
status: draft | review | approved | verified | deprecated | obsolete | frozen
priority: must | should | could       # MoSCoW; SAFe — через WSJF (BR-specific)

# Provenance (conditional, standard/07 §7.4.1): ADAPT реактивен.
# Findings present → source.adapt + adapt-section mandatory. No findings →
adversarial-review-ref mandatory.
# source.tz-section — обязательно всегда.
source:
  adapt: "ADAPT-NNN"                  # conditional
  adapt-section: "Forward §N.N"       # mandatory если adapt present
  tz-section: "§N.N"                  # обязательно всегда
  adversarial-review-ref: "<ссылка>"  # mandatory когда adapt omitted
  document-ref: "<ссылка>"            # pinned ревизия source-документа

# Hierarchy
parent: { id: "<parent-id>", ref: "<ссылка>" }   # id required для SR (→BR), TR
(→SR); optional для BR
children: []                          # auto-derived
```

```
# Связь с SPEC (граф)
constrained-by: []                    # SR → SPEC-* (типизированные рёбра)
implements-spec: []                   # TR → SPEC-*
depends-on: []                        # между SPEC

# Verification
verified-by: []                       # auto-derived; TC IDs
verifies-business-goal: ""            # optional

# AI provenance (RENAR-4+ обязательно для approved)
ai-provenance:
  generated-by: "<vendor>-<model>@<date>"
  prompt-template: "<template-path>@<version>"
  context-tokens: integer
  output-tokens: integer
  generation-time-ms: integer
  generated-at: "<ISO-8601>"
  human-edits: boolean                # true required для approved

# AI cost budget (optional)
ai-budget: { context-tokens-target: integer, context-tokens-actual: integer,
output-tokens-target: integer, output-tokens-actual: integer, generation-time-
target-ms: integer }

# Замена + schema versioning
replaces: "<old-id>"
replaced-by: "<new-id>"
deprecated-date: "<ISO date>"
schema-version: "1.0"
```

2. BR — Business Requirement

```
# Extends common §1, дополнительно:

level: system | subsystem            # BR на уровне модуля запрещён (standard/06
§6.4)
scope: { system: "<system-id>", subsystem: "<subsystem-id>" }

# Межуровневая связь BR подсистемы → BR системы (standard/06 §6.8.2)
implements:                            # массив; substrate-agnostic ссылка
  - { id: BR-NN, scope: { system: "<system-id>" }, rationale: "<short>" }   #
rationale опционально
implemented-by: []                     # auto-derived (обратное ребро; не пишется
автором)

business-context:
  stakeholder: "<role>"
  business-goal: "<short statement>"
  kpi-impact:
    - { kpi: "<name>", direction: increase|decrease, target: "<measurable>" }
```

```
# business-outcome — required для QG-4
business-outcome:
  measurement-type: kpi | survey | observation | usage
  kpi-name: "<KPI>"
  measurement-method: "<how>"
  baseline-value: number
  baseline-measured-at: "<ISO date>"
  target-value: number
  target-met-by: "<ISO date>"
  current-value: { value: number, measured-at: "<ISO date>", achievement: "
<percent>" }

prioritization: { framework: WSJF|RICE|MoSCoW, wsjf-score: number, prioritized-
at: "<ISO date>", prioritized-by: "<role>" }

data-classification:
  contains-pii: boolean
  contains-financial: boolean
  contains-health: boolean
  contains-children-data: boolean
  retention-days: integer
  data-residency: ["RU" | "EU" | "US" | ...]

compliance:
  - { standard: "ISO 27001:2022", control: "<id>", rationale: "..." }
  - { standard: "GDPR", article: "Art.NN" }
  - { standard: "ФЗ-152", article: "ст.NN" }

ai-act: { risk-class: prohibited|high|limited|minimal, rationale: "<reason>",
high-risk-domain: boolean }
```

Поля `implements[]` / `implemented-by[]` — нормативные правила

Правило	Уровень
`implements[]` обязателен при `level: subsystem` И когда родительская система имеет ≥1 approved BR	recommended v1.0; обязательно v1.1+
Target BR обязан быть в статусе `approved` или выше на момент approve данного BR	hook-enforced
Cycle detection: цепочка `implements` не должна образовывать циклов	hook-enforced
`implements[]` — **не** parent-edge; запрет множественных parents (standard/06 §6.8.3) распространяется на SR/TR, не на BR	normative
Deprecate target BR → cascade-warning для всех `implemented-by` (не cascade-deprecate)	hook-enforced
Cross-substrate синтаксис: `id + scope.system` не зависит от носителя	normative
Cardinality: array (0..N)	normative

Гейт обеспечения соблюдения — `scripts/check-implements-edge.js`. Поле `implemented-by` — auto-derived; ручная запись запрещена.

3. SR — System Requirement

```
# Extends common §1, дополнительно:

parent:
  id: "BR-NN"                          # required

# Источник ADAPT — через канонические source.adapt / source.adapt-section (§1).
# Отдельного поля derived-from-adapt нет (standard/06 §6.6.2).

constrained-by:                        # типизированные рёбра к SPEC-*
  - "SPEC-UI-NN"
  - "SPEC-API-NN"
  - "SPEC-DATA-NN"
  - "SPEC-SEC-NN"

quality-characteristic:                # ISO/IEC 25010:2023 (9 характеристик;
interaction-capability ← usability, flexibility ← portability в 25010:2011;
safety — новая в 25010:2023)
  - functional-suitability | performance-efficiency | compatibility |
interaction-capability | reliability | security | maintainability | flexibility |
safety

# Inherited from parent BR (если применимо): data-classification, compliance, ai-
act
```

4. TR — Task Requirement

```
# Extends common §1, дополнительно:

parent:
  id: "SR-NN"                          # required

implements-spec: []                    # SPEC-* реализуемые этой задачей
estimated-effort: "<short statement>" # optional, free-form
```

5. SPEC-* common schema

Все 9 типов SPEC делят общую структуру (§1) + следующие SPEC-specific поля:

```
type: SPEC-ARCH | SPEC-API | SPEC-DATA | SPEC-INT | SPEC-PROC | SPEC-UI | SPEC-AI
| SPEC-SEC | SPEC-OPS

referenced-by: []                    # auto-derived
depends-on: []                       # SPEC от которых этот зависит
compliance-refs: []                  # ISO / GDPR / ФЗ-152 / AI Act / NIST AI
RMF
```

Обязательные разделы body: `## Назначение`, `## Scope`, `## <Type-specific sections — см. §6>`, `## Связь с требованиями`, `## Связь с другими SPEC`, `## Verification`, `## Open questions`.

---

6. SPEC type-specific extensions

Type-specific поля для 9 типов SPEC. Industry references — в `standard/14`. Legacy замены: `UIC` → SPEC-UI, `AIC` → SPEC-AI, `INT-SR` → SPEC-INT.

6.1 SPEC-ARCH, SPEC-API, SPEC-DATA, SPEC-INT, SPEC-PROC

```
# SPEC-ARCH
arch-style: monolith | microservices | modular-monolith | serverless | hybrid
deployment-model: cloud | on-prem | hybrid | edge
tech-stack: { languages: [], frameworks: [], data-stores: [], message-brokers: []
}
quality-attributes: [{ name: latency, target: "p95 < 200ms" }, { name:
availability, target: "99.9%" }]

# SPEC-API
api-style: rest | graphql | grpc | websocket | async-events
api-version: "v1.2.0"
versioning-strategy: url-path | header | query-param | content-negotiation
authentication: bearer-jwt | api-key | oauth2 | mtls | none
rate-limits: [{ endpoint: "*", limit: "1000/min/key" }]
contract-file: { format: openapi-3.1 | asyncapi-2.6 | proto3, location:
"contracts/<name>.yaml" }

# SPEC-DATA
data-style: relational | document | graph | columnar | hybrid
storage-engine: postgresql | mysql | couchdb | mongodb | clickhouse | ...
schema-version: "1.4.0"
pii-classification: [{ entity: User, fields: [email, phone], level: PII-high }]
retention-policies: [{ entity: Order, period: "7 years", basis: "tax law" }]
migration-strategy: forward-only | reversible | dual-write

# SPEC-INT
integration-pattern: request-response | event-driven | message-queue | webhook |
file-transfer
```

```
direction: outbound | inbound | bidirectional
counterparty: { system: "<external-name>", contract-owner: "<team-or-vendor>",
contract-ref: "<external-spec-url>" }
sla: { availability: "99.5%", latency-p95: "500ms", fallback: "queue + retry;
manual reconciliation after 24h" }
idempotency: guaranteed | best-effort | none

# SPEC-PROC
process-style: bpmn | state-machine | saga | choreography | orchestration
state-count: integer
participants: [{ role: customer, system: client-portal }, { role: agent, system:
back-office }]
sla: { end-to-end: "2 business hours" }
compensation: defined | not-applicable | manual
```

6.2 SPEC-UI, SPEC-AI, SPEC-SEC, SPEC-OPS

```
# SPEC-UI
ui-platform: web | mobile-ios | mobile-android | desktop | tv | embedded
target-users: [{ role: end-customer, persona: "ADAPT-NNN §X.Y" }]
design-system: "<reference-or-internal>"
accessibility-level: WCAG-A | WCAG-AA | WCAG-AAA
i18n: required | not-required
mockup-links: [{ tool: figma, url: "<link>", version: "v3" }]
baseline-images: ["ai-concepts/baselines/SPEC-UI-NN-screen-01.png"]

# SPEC-AI (judge-model.vendor ≠ production-model.vendor — нормативно)
ai-pattern: rag | fine-tuning | prompt-engineering | tool-use | multi-agent |
embedding-only
production-model: { vendor: anthropic|openai|google|local, model: "<name>",
version: "<exact>" }
judge-model: { vendor: "<different-vendor>", model: "<different-model>" }
context-strategy: { embedding-model: "<model>", chunk-size: integer, chunk-
overlap: integer, vector-store: pinecone|weaviate|pgvector|qdrant }
eval-strategy: { metric: accuracy|f1|rouge|custom-rubric, threshold: number,
baseline-dataset: "<path>" }
cost-budget: { tokens-per-request-target: integer, tokens-per-request-ceiling:
integer, monthly-budget-usd: number }

# SPEC-SEC
security-domains: [authentication, authorization, data-protection, audit,
secrets-management]
auth-model: { authn: jwt-bearer|oauth2-pkce|mtls|passkey, authz: rbac|abac|relbac
}
data-classification: [{ class: PII-high, fields: [...] }, { class: PCI, fields:
[...] }]
threat-model-method: STRIDE | PASTA | OCTAVE
compliance: [ISO-27001, GDPR, Ф3-152, PCI-DSS-4]

# SPEC-OPS
deployment-style: kubernetes | vm | serverless | docker-compose | bare-metal
```

```
environments:
  - { name: dev, purpose: development, scale: minimal }
  - { name: staging, purpose: integration-testing, scale: half-prod }
  - { name: prod, purpose: production, scale: full }
slo: { availability: "99.9%", error-budget-month: "43m", latency-p95: "300ms" }
observability: { logs: elastic|loki|cloudwatch, metrics:
prometheus|datadog|cloudwatch, traces: jaeger|tempo|x-ray }
disaster-recovery: { rto: "<duration>", rpo: "<duration>" }
```

7. ADAPT schema

ADAPT — отдельный артефакт (`standard/07`). Реактивный: существует только при findings от состязательного обзора ТЗ (§7.4.1). Вердикт «no findings» — ADAPT не создаётся, фиксируется через `<artifact>.source.adversarial-review-ref` .

```
# Identity
id: ADAPT-NNN
title: "Адаптация ТЗ <name>"
type: ADAPT
trigger-stage: import-tz | decompose-br | decompose-sr | spec | tc   # стадия-
триггер (standard/07 §7.4.1.4)

# Source
source-tz: { id: TZ-YYYY-NNN, signed-date: "<ISO-date>", signed-by-client: "
<name-role>", document-ref: "<ссылка>" }
parent-adapt: { id: ADAPT-NNN, delta-tz: TZ-YYYY-NNN }   # для delta-ADAPT

# Supersession (standard/07 §7.6.4) — только для superseding-ADAPT
supersedes: ADAPT-MMM                                    # ссылка на
дезавуируемый ADAPT
superseded-by: ADAPT-NNN                                 # auto-derived; на
дезавуируемом
supersession-rationale: "<противоречащее BR/SR/SPEC + источник>"   # mandatory
если supersedes присутствует

# Lifecycle (подмножество §1: ADAPT не использует verified/deprecated; superseded
— терминальное при дезавуировании)
status: draft | review | client-ready | answered | approved | frozen | superseded
| obsolete
created: "<ISO-date>"
last-updated: "<ISO-date>"

# Approval (required для approved)
approval:
  client-signature: { signed-by: "<name>", role: "<role>", organization: "
<client-org>", signed-at: "<ISO-datetime>", signature-ref: "<ссылка>" }
  architect-signature: { signed-by: "<name>", role: architect, signed-at: "<ISO-
datetime>" }

# Auto-derived
```

```
generates-requirements: []
generates-specs: []
open-questions-count: integer          # должен быть 0 для approved
resolved-questions-count: integer

# AI provenance
ai-provenance: { generated-by: "<vendor>-<model>@<date>", prompt-template: "
<template-path>@<version>", context-tokens: integer, output-tokens: integer,
human-edits: boolean }
```

Backward записи внутри body:

```
id: B-NNN
category: contradiction | gap | hidden-assumption | feasibility | regulatory |
terminology | scope
status: open | asked-to-client | answered | resolved | frozen
tz-section: "§N.N"
description: "..."
asked-to-client: "<ISO-date>"
client-answer:
  signed-by: "<name>"
  signed-at: "<ISO-datetime>"
  channel: email | docusign | zoom-transcript | written-letter
  text: "..."
resolution: "..."                      # как ответ интегрирован в Forward
```

8. TC — Test Case

```
# Identity
id: "TC-NN[.N]"
title: "<descriptive>"
type: TC
slug: "<kebab-case>"

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

# Scope
level: system | subsystem | module
scope: { system: "<system-id>", subsystem: "<subsystem-id>", module: "<module-
id>" }

# Lifecycle
status: draft | ready | passing | failing | obsolete

# Verification mapping (≥1)
verifies:
```

```
  - { id: "<requirement-id>", ref: "<ссылка>", requirement-version: "<version-ref>" }   # V5 pinning (standard/03 §3.3.5)

# Pair link (mandatory если negative=false и существует парный)
paired-with: ["<TC-id>"]

# Automation
automation:
  status: automated | manual-pending
  location: "<path-to-implementation>"      # mandatory если automated
  runner: pytest | jest | go-test | playwright | vlm-judge | ragas | pact | other
  manual-pending-until: "<ISO date>"        # mandatory если manual-pending
  manual-pending-reason: "<why>"

# Execution (mandatory если tc-type=ux | eval; judge.vendor ≠ production model vendor — см. §6.2 SPEC-AI, §9)
judge: { vendor: "<provider>", model: "<model-id>", prompt-template: "<template-path>@<version>" }
baseline: { artifact: "<pointer>", perceptual-diff-threshold: float, metric-thresholds: {} }

# Last run (auto-managed; bot-only)
last-run:
  date: "<ISO timestamp>"
  result: pass | fail | skipped | n/a
  runner-id: "<runner@version>"
  run-ref: "<ссылка>"
  requirement-version: "<version-ref>"
  judge-report: "<for ux/eval>"

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

# Inherited
ai-provenance: { ... }                # см. §1
```

9. Validation rules (cross-field)

Правила, не выражаемые в чистой JSON Schema; требуют custom validator. Колонка «Формальная проверка» даёт исполнимый предикат или ссылку на готовый KG-запрос (reference/05 §5/§6).

Правило	Описание	Формальная проверка
ID неизменяем	При изменении файла поле `id` не меняется.	`diff(prev.id, curr.id) == ∅`
`parent` exists	Для SR — parent BR существует и в статусе ≥ `approved`.	`status(BR[SR.parent.id]) ≥ approved`; orphans — 05 §6.1

`source.adapt` approved	Для BR/SR/SPEC — ADAPT в `source.adapt` в статусе `approved` / `frozen`.	`status(ADAPT[art.source.adapt]) ∈ {approved, frozen}`
`verified-by` consistency	TC в `verified-by` имеют `verifies[].id` = этот артефакт.	`∀ tc ∈ art.verified-by: art.id ∈ tc.verifies[].id`
`requirement-version` lock	`TC.last-run.requirement-version` = `verifies[].requirement-version`.	`tc.last-run.requirement-version == tc.verifies[].requirement-version`; stale — 05 §4.4
`source.adapt` для BR/SR/SPEC (conditional)	Канонический источник ADAPT при наличии findings; при вердикте «no findings» — `source.adversarial-review-ref` (standard/07 §7.4.1). TR — через parent SR (standard/06 §6.6.2).	`art.type ∈ {BR, SR, SPEC-*} ⇒ art.source.adapt ≠ null ∨ art.source.adversarial-review-ref ≠ null`
SPEC-AI requires ai-act	Для AI-артефакта `ai-act.risk-class` обязательно.	`art.type == SPEC-AI ⇒ art.ai-act.risk-class ≠ null`
Data residency consistency	RU в `SR.data-classification.data-residency` ⇒ то же в parent BR.	`'RU' ∈ SR.…data-residency ⇒ 'RU' ∈ BR[SR.parent].…data-residency`
Compliance hierarchy	`SR.compliance ⊆ parent BR.compliance` (или явный justification).	`SR.compliance ⊆ BR[parent].compliance ∨ exists(extension-justification)`
TC `automated` requires location	`automation.status: automated` ⇒ `automation.location` непустое.	`tc.automation.status == 'automated' ⇒ tc.automation.location ≠ ''`
Negative TC обязателен	На каждое нормативное утверждение — TC с `negative: true`.	`∀ assertion ∈ art: ∃ tc(negative: true)` (standard/09 §9.7)
ADAPT open-questions == 0 for approved	Approval блокируется при `open` / `asked-to-client` backward.	`adapt.status == approved ⇒ count(backward[status ∈ {open, asked-to-client}]) == 0` (05 §4.7)
Дезавуирование корректно	`supersedes` ⇒ непустой `supersession-rationale` и симметричный `superseded-by` на цели; нет висячих `source.adapt` на `superseded` (standard/07 §7.6.4, standard/10 §10.8.5).	`adapt.supersedes ≠ null ⇒ adapt.supersession-rationale ≠ '' ∧ ADAPT[adapt.supersedes].superseded-by == adapt.id`; `∄ art: art.source.adapt = X ∧ status(ADAPT[X]) == superseded`
Judge isolation (SPEC-AI)	`judge.vendor ≠ production-model.vendor`.	`tc.judge.vendor ≠ SPEC-AI[tc.verifies].production-model.vendor`

SPEC depends-on acyclic	Граф `depends-on` между SPEC — DAG.	cypher cycle-detection (05 §4.6): rows ≠ ∅ ⇒ нарушение

10. Изоморфизм носителя

Отображение для git (YAML frontmatter) ↔ document-oriented store (JSON document):

Поле (canonical)	git (YAML frontmatter)	Document store (JSON doc)
`id`	`id`	`_id = <project>:<doc-type>:<slug>`, поле `slug`
`type: BR`	`type: BR`	`level: "business"`
`type: SPEC-API`	`type: SPEC-API`	`doc_type: "spec_api"`
`parent.id`	`parent.id`	`parent`
`children`	(auto-derived)	`children` (auto-derived)
`status`	`status`	`status`
`priority`	`priority`	`priority`
`source.adapt`	`source.adapt`	`created_from_adapt`
`constrained-by[]`	`constrained-by`	`constrained_by` (subdoc array)
`verified-by[]`	(auto-derived)	`linked_tests`
`ai-provenance.*`	nested object	nested subdocument
`compliance`	`compliance` array	`compliance` subdoc
`data-classification`	nested object	nested subdoc
`business-outcome`	nested object	nested subdoc
`replaces` / `replaced-by`	string ID	`replaces` / `replaced_by`

Нативные для носителя имена полей могут отличаться, но **семантика и invariants сохраняются** через capabilities V1-V6 (01-glossary.md §27).

11. Schema versioning

Каждый артефакт имеет поле `schema-version` (semver). При несовпадении версии файла и текущей схемы validator предлагает migration.

Изменение	Bump
Новое необязательное поле	minor (1.0 → 1.1)

Новое обязательное поле	major (1.0 → 2.0) + migration script
Удаление поля	major + migration script
Изменение enum	minor если добавление, major если удаление
Переименование поля	major + migration script

Текущая версия schemas: 1.0-draft.

12. JSON Schema fragment example (BR)

Ключевые patterns (полная BR-схема — `reference/schemas/br.json`, планируется):

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "https://renar.tech/schemas/br.json",
  "type": "object",
  "required": ["id", "title", "type", "status", "priority", "source", "ai-
provenance"],
  "properties": {
    "id":     { "type": "string", "pattern": "^BR-[0-9]{2}(\\.[0-9]+)?$" },
    "title":  { "type": "string", "minLength": 5, "maxLength": 100 },
    "type":   { "const": "BR" },
    "status": { "enum": ["draft", "review", "approved", "verified", "deprecated",
"obsolete"] },
    "priority": { "enum": ["must", "should", "could"] },
    "source": { "type": "object", "required": ["tz-section"], "properties": {
"adapt": { "pattern": "^ADAPT-[0-9]{3}(-delta-[0-9]+)?$" } } },
    "ai-provenance": { "required": ["generated-by", "generated-at"],
"properties": { "generated-by": { "pattern": "^[a-z]+-[a-z0-9-]+@[0-9]{4}-[0-9]
{2}-[0-9]{2}$" } } }
  }
}
```

Аналогичные JSON-схемы для SR/TR/SPEC-*/TC/ADAPT — в `reference/schemas/` (планируется).

Schemas reference RENAR 1.0-draft — см. также 01-glossary.md, `standard/06` - `09` для нормативных определений.

AI Risk Register для RENAR-проектов

> ***Назначение:*** *реестр AI-специфичных рисков для проектов, использующих RENAR (где AI генерирует требования, спецификации и тесты). Основан на ISO/IEC 23894:2023 (AI Risk Management, Annex A risk sources + Clause 6 process по ISO 31000) и NIST AI RMF 1.0. Нормативные mitigation hooks — standard/09 §9.4, standard/07 §7.10.*

Не подменяет общий security risk register организации. AI-риски — отдельный класс из-за специфики генерации и непредсказуемости моделей.

1. Структура реестра

Каждый риск имеет:

```
id: AIR-NN                              # immutable
name: "<short name>"
category: hallucination | injection | drift | bias | sgnl-failure | data-quality
| adversarial | privacy
# enum-значение — часть до скобки; уточнитель в скобках опционален (напр. "sgnl-
failure (process)")
severity: critical | high | medium | low
likelihood: high | medium | low
iso-23894-ref: "§N.N"
nist-rmf-function: govern | map | measure | manage
mitigations:
  - { mechanism: "<description>", enforced-by: "<who/what>", automated: true |
false }
status: active | mitigated | accepted | monitoring | out-of-scope
owner: "@role"
last-reviewed: "YYYY-MM-DD"
related: ["<core rule N>", "<standard chapter>", "<other AIR-NN>"]
```

Список AIR-01..AIR-14 закрыт; новые риски — только через изменение полного RENAR Standard.

Category ↔ NIST AI RMF trustworthiness characteristics (для проверяемости заявления «основан на NIST AI RMF»):

category	NIST AI RMF trustworthiness characteristic
`hallucination` / `data-quality`	Valid & Reliable
`injection` / `adversarial`	Secure & Resilient
`drift`	Valid & Reliable; Safe
`bias`	Fair — with Harmful Bias Managed
`sgnl-failure`	Safe; Accountable & Transparent

`privacy`	Privacy-Enhanced

> ***Ссылки ISO/IEC 23894:2023.*** *Категории AI-рисков в 23894:2023 расположены в* ***Annex A*** *(risk sources); Clause 6 описывает процесс риск-менеджмента (по ISO 31000). Дескрипторы «Annex A — …» в реестре — risk-source labels; точное сопоставление с пунктами Annex A подлежит сверке при формальном claim.*

2. Реестр AIR-01..AIR-14

Метаданные всех 14 рисков (Sev=Severity, Like=Likelihood):

AIR	Name	Category	Sev	Like	ISO 23894 (Annex A)	NIST RMF	Status
01	Hallucination в AI-генерируемых требованиях	hallucination	**High**	High	Output reliability	Measure	active → mitigate зрелых RENAR levels
02	Prompt injection через T3 от клиента	injection	**High**	Low-Medium	Adversarial inputs	Manage	monitori
03	Model drift / version change	drift	Medium	High	Model жизненный цикл	Manage	monitori
04	Bias в AI-генерации требований	bias	Medium	Medium	Fairness	Measure	active
05	Single-model failure (no diversity)	sgnl-failure	Medium	High	Single point of failure	Manage	mitigate при пол pipeline
06	Test fitting / зеленение тестов	sgnl-failure	**High**	Medium	Verification integrity	Measure	mitigate при выборо провер
07	Hallucinated citations	hallucination	Medium-High	Medium	Output reliability	Measure	monitori
08	Adversarial inputs в clients data (runtime)	adversarial	**High**	Low	Adversarial inputs	Manage	out-of-scope (applicat level)
09	Privacy leakage через AI logs	privacy	**High**	Medium	Privacy	Govern	active
10	Knowledge graph poisoning	data-quality	Medium	Low	Data integrity	Map	monitori

11	Reconciliation false-positive overload	sgnl-failure (process)	Low-Medium	Medium	Verification integrity	Manage	monitori
12	Cost runaway (uncontrolled AI spend)	sgnl-failure (operational)	Medium	Medium	Cost governance	Manage	active
13	Стейкхолдер не понимает AI-сгенерированные требования	data-quality (UX)	Medium	Medium	Transparency	Govern	active
14	Vendor lock-in to specific LLM provider	sgnl-failure (operational)	Medium	Low-Medium	Vendor risk	Govern	monitori

Описание + воздействие + mitigations — ниже.

AIR-01. Hallucination в AI-генерируемых требованиях

AI-агент при генерации BR/SR/SPEC может «дописать от себя» утверждения, которых нет в ADAPT или ТЗ. Воздействие: scope creep, dispute на acceptance, фичи которые не требовались клиентом. **Меры смягчения:** source citation (RENAR Core Правило 1 — каждое утверждение ссылается на ADAPT §N); состязательный обзор (критик-модель другого vendor); метрика Hallucination Rate ≤1% на зрелых уровнях RENAR.

AIR-02. Prompt injection через ТЗ от клиента

Злонамеренный клиент может вставить в ТЗ скрытые инструкции для AI («ignore previous instructions and …»). Воздействие: утечка данных, вредоносные изменения в требованиях, нарушение security policy. **Меры смягчения:** sandboxing AI-агента при импорте (модель работает с ТЗ как с пассивными данными, явно в system prompt); input gateway проверяет на known injection patterns; suspicious pattern → escalation, не auto-process.

AIR-03. Model drift / version change

Anthropic / OpenAI / Google обновляют модели — тот же prompt с тем же ТЗ через 6 месяцев может дать другой output. Воздействие: inconsistency между требованиями в одном проекте; невозможность точно воспроизвести генерацию старого артефакта. **Меры смягчения:** model versioning в `ai-provenance.generated-by` (точная версия + дата); eval-тесты для SPEC-AI прогоняются при смене модели; при регенерации — diff против старой версии и оценка изменений.

AIR-04. Bias в AI-генерации требований

Модель имеет training-data bias — при генерации BR может игнорировать stakeholders определённых групп (accessibility users, non-English locales, специфичные регуляторики). Воздействие: требования не покрывают весь spectrum пользователей; продукт дискриминационный или non-compliant. **Меры смягчения:** multi-model agreement для `priority=must` BR (разные модели — разные biases); карта заинтересованных сторон обязательна в BR (explicit перечисление); состязательный критик с prompt «check for missing stakeholders / accessibility considerations».

AIR-05. Single-model failure (no diversity)

Если все артефакты генерируются одной моделью, её ошибки систематически проникают. «Галлюцинирует» определённый паттерн — все требования это унаследуют. Воздействие: систематическое искажение требований по проекту. **Меры смягчения:** multi-model для `priority=must` BR; состязательный критик с другой моделью; изоляция judge-модели от production-модели в eval-тестах (SPEC-AI: `judge-model.vendor ≠ production-model.vendor`, см. 02-schemas.md §6.2).

AIR-06. Test fitting / зеленение тестов

AI-агент имеет тривиальный путь зеленения failing-теста — ослабить Pass-критерий. Это проходит code review, поскольку «тест зелёный». Воздействие: false confidence; defects проходят в production. **Меры смягчения:** маркер `[test-spec-change]` обязателен для изменения Pass/Fail (отдельный approval); выборочная проверка 5 случайных passing TC раз в спринт (RENAR Core Правило 5); метрика Test-fitting drift rate (отдельная от обычных metrics).

AIR-07. Hallucinated citations

AI-агент пишет citation `[TZ-2026-001 §4 line 142]`, но в реальном ТЗ §4 line 142 — про другое. Citation выглядит как свидетельство, но свидетельство ложное. Воздействие: source citation становится фикцией; цепочка прослеживаемости рвётся при аудите. **Меры смягчения:** citation validator hook (парсит citation, открывает указанный документ, проверяет соответствие); pre-commit/pre-approval блокировка при невалидной citation.

AIR-08. Adversarial inputs в clients data (runtime)

Клиент отправляет данные (через формы, API), специально сконструированные для манипуляции AI-компонент в runtime (не на этапе генерации требований). Воздействие: аналогично AIR-02, но в production runtime. **Меры смягчения:** input sanitization на API gateway; constrained generation (structured outputs only); rate limiting per user. **Status: out-of-scope** — application-level security; RENAR требует SR-уровень покрытия (SPEC-SEC threat model), но runtime защита — задача реализации, не нормирования требований.

AIR-09. Privacy leakage через AI logs

AI-агент при генерации артефакта имеет в контексте PII (из ТЗ или интервью). Логи генерации (tool events, audit records) могут хранить эти PII. Воздействие: PII попадают в логи, в `ai-provenance`, в training data (если используется). **Меры смягчения:** PII redaction в промптах перед отправкой в LLM; `data-classification` tracking; disable training on conversations (Anthropic/OpenAI privacy settings, DPA); TTL на event logs с PII.

AIR-10. Knowledge graph poisoning

Если KG используется как primary search для AI-агентов, incorrect edge может «отравить» все последующие AI-запросы, опирающиеся на этот граф. Воздействие: AI генерирует требования на основе wrong context, систематически. **Меры смягчения:** KG derived от frontmatter, не редактируется

напрямую (см. 05-knowledge-graph-schema.md); CI-валидация графа на каждое изменение (нет circular dependencies, no orphan approved); reconciliation-агент проверяет integrity еженедельно.

AIR-11. Reconciliation false-positive overload

Reconciliation-агент при слишком чувствительных правилах генерирует много false-positive находок. Архитектор начинает их игнорировать → реальные находки тонут. Воздействие: reconciliation теряет ценность, дисциплина не масштабируется. **Меры смягчения:** tunable thresholds в проектной конфигурации; метрика Reconciliation Findings/Week (если растёт без real issues — re-calibration); архитектор может отклонять находки с обоснованием — feedback для tuning prompt агента.

AIR-12. Cost runaway (uncontrolled AI spend)

Без budget tracking AI-генерация (особенно с multi-model, состязательный, eval) может потреблять токены непропорционально размеру проекта. Воздействие: финансовые потери; нерентабельность практики. **Меры смягчения:** `ai-budget` field в frontmatter (target + actual); aggregated cost metric на уровне проекта; cap на проект; alarm при approached; recommendation engine «Sonnet/Haiku для рутины, Opus только для `priority=must` BR».

AIR-13. Стейкхолдер не понимает AI-сгенерированные требования

AI генерирует SR в техническом стиле; клиент / нетехническая заинтересованная сторона при рецензировании не понимает → утверждение становится формальностью. Воздействие: QG-ADAPT-approve / QG-4 acceptance теряет смысл; dispute rate at acceptance растёт. **Меры смягчения:** style guide (04-ai-style-guide.md); BR в business language (технологии — в SPEC-*, не в BR); human-readable summary в каждом BR/SR — короткая секция, понятная без технического background.

AIR-14. Vendor lock-in to specific LLM provider

Все промпты оптимизированы под конкретного провайдера (Anthropic Claude). Если provider меняет pricing/availability — переход требует переписывания всех промптов. Воздействие: operational risk, costs, business continuity. **Меры смягчения:** provider-agnostic prompts где возможно (избегать vendor-specific tool syntax); multi-model уже enforced для `priority=must` (Принцип 4) — гарантирует второй провайдер в pipeline; periodic test runs на резервном провайдере.

---

3. Risk matrix

```
Severity / Likelihood
         │  Low     Medium        High
─────────┼──────────────────────────────────
High     │  AIR-08  AIR-04, 06    AIR-01, 03
Medium   │  AIR-10  AIR-11, 13    AIR-07, 09, 14
Low      │  —       AIR-12        AIR-02, 05
```

Critical и High риски в top-right квадранте — приоритет mitigation.

---

4. Mitigation matrix

Какие mitigations покрывают какие риски (компенсирующие механизмы: одиночный mitigation редко достаточен; высокие риски требуют ≥ 2 независимых механизмов):

Mitigation	Покрывает риски
Source citation (Core Правило 1)	AIR-01, AIR-07
Состязательный обзор (другая модель)	AIR-01, AIR-04, AIR-05
Выборочная проверка passing TC (Core Правило 5)	AIR-06
Multi-model для `priority=must`	AIR-04, AIR-05, AIR-14
Judge isolation в SPEC-AI	AIR-05
AI-происхождение (model + version + date)	AIR-03, AIR-09
Citation validator hook	AIR-07
Input sandbox / sanitization	AIR-02, AIR-08
PII redaction + DPA	AIR-09
KG validation in CI	AIR-10
Reconciliation tunable thresholds	AIR-11
`ai-budget` field + project cap	AIR-12
Style guide + business-language BR	AIR-13

5. Operational governance

Периодичность рецензирования. Monthly: AIR-01, AIR-02, AIR-06, AIR-07, AIR-09 (high-impact runtime risks). Quarterly: остальные. On-incident: любой риск, в который произошёл инцидент → root cause → mitigation.

Owner. Default — Архитектор проекта. Для AI-специфичных рисков — AI Governance Lead (если есть в организации).

Storage. Risk register проекта — отдельный артефакт:

```
<project>.req/
  governance/
    ai-risk-register.md          # snapshot этого reference + project-specific
notes
    review-log.md                # история ревью с датами и подписями
```

На носителе, не поддерживающем директории — эквивалент namespacing.

Reconciliation-агент. Еженедельный run обновляет `status` и `last-reviewed` поля каждого AIR. Если статус меняется (e.g., monitoring → active) — alert архитектору.

6. Связь со стандартом

AIR	RENAR Core / Standard
AIR-01, 07	Core Правило 1 (ADAPT перед SR) + Standard ch.5
AIR-04, 13	Standard ch.4 (роли) + style guide §04
AIR-05	Standard ch.13 (AI generation) + SPEC-AI judge isolation
AIR-06	Core Правило 4 + Правило 5 + QG-2 Verification Gate
AIR-09	Standard ch.11 compliance + SPEC-SEC
AIR-12	Standard ch.13 (cost governance)

7. Что НЕ покрывает risk register

Этот реестр focuses на AI-специфичные риски процесса RENAR. **Не подменяет:** общий security risk register организации (ISO 27001); compliance risk register (06-compliance.md); application-level threat model конкретного проекта (SPEC-SEC). Reg-обязательные требования регуляторов (AI Act high-risk, ФЗ-152) — отдельные artifacts; этот register — operational tier.

AI Risk Register RENAR 1.0-draft — renar.tech

Схема графа знаний

> ***Назначение:*** *формальная схема графа знаний (KG) для RENAR-проектов — узлы, грани, properties, query patterns. KG — **derived view** от RENAR-артефактов для семантических запросов AI-агентов и reconciliation. Machine-readable edge types — §3; нормативные поля связей — standard/06 §6.10, standard/08.*

KG — **не источник правды**. Источник истины — артефакты (ADAPT / BR / SR / SPEC / TC) на носителе. Граф derived from frontmatter и не редактируется напрямую. Если граф противоречит артефактам → rebuild графа, не правка артефактов.

1. Use cases

Без KG AI-агент имеет плоский поиск (FTS5/RAG + `parent` / `children` direct hops + keyword grep) — синтаксический контекст. Граф добавляет семантический:

Запрос	Без графа	С графом
Все BR, влияющие на Sales Cycle KPI	Grep + парсить frontmatter	Один Cypher-запрос
При изменении SR-05 — какие задачи и SPEC затронуты	Сканировать все ссылки	Single graph traversal
Какие SPEC-AI требуют high-risk classification по AI Act	Вручную	One query
Карта заинтересованных сторон для проекта	Несколько запросов	Single subgraph extraction
Cross-project dependencies через SPEC-INT	Federation API calls	Federated graph query
Стало ли требование SR-12 деградировать	Manual analysis	Trend analytic on node properties
Stale TC (verifies SR с обновлённой версией, last-run на старой)	Manual check	One query

2. Node types (закрытый список)

Type	Источник	Identity
`Requirement`	BR / SR / TR артефакты	`<id>`
`Specification`	SPEC-* артефакты (9 типов)	`<spec-id>`
`ADAPT`	ADAPT-NNN артефакты	`<adapt-id>`
`BackwardFinding`	B-NNN записи внутри ADAPT	`<adapt-id>:B-NNN`

`TestCase`	TC артефакты (вкл. `tc-type: contract`)	`<tc-id>`
`WorkOrder`	T3 и delta-T3	`<tz-id>`
`Stakeholder`	frontmatter `business-context.stakeholder`	`<stakeholder-id>`
`BusinessGoal`	frontmatter `business-context.business-goal` (deduplicated)	`<goal-id>`
`KPI`	frontmatter `business-outcome.kpi-name`	`<kpi-id>`
`Task`	TR / runtime task store	`<task-id>`
`CodeArtifact`	TC `automation.location`, code commits	`<repo>:<path>:<symbol>`
`Decision`	Architectural decision records	`<decision-id>`
`DeadEnd`	Failed approaches (опционально)	`<deadend-id>`
`RiskItem`	AI Risk Register entry	`AIR-NN`
`Compliance`	Compliance standard reference	`<std>:<control>`
`Template`	Requirements library template	`<template-id>`

Список закрыт; новые типы узлов — только через изменение полного RENAR Standard.

3. Edge types (закрытый список)

3.1 Иерархия и derivation

Edge	From	To	Семантика
`parent`	Requirement	Requirement	A.parent = B → B is parent of A (BR → SR → TR)
`derived-from-adapt`	Requirement / Specification	ADAPT	Артефакт выведен из approved ADAPT section
`derived-from-template`	Requirement / Specification	Template	Шаблон (с version pin)
`replaces`	Requirement / Specification	Requirement / Specification	new replaces old (deprecated)
`supersedes`	Requirement / Specification	Requirement / Specification	strictly newer version (post-delta)
`parent-adapt`	ADAPT	ADAPT	delta-ADAPT → main ADAPT

3.2 ADAPT specifics

Edge	From	To	Семантика
`from-tz`	ADAPT	WorkOrder	source-tz pointer
`parent-adapt`	ADAPT	ADAPT	delta-ADAPT → корневой (`parent-adapt`)
`supersedes`	ADAPT	ADAPT	дезавуирующий ADAPT → дезавуируемый; обратное `superseded-by` (standard/07 §7.6.4); цель переходит в `superseded`
`contains-backward`	ADAPT	BackwardFinding	B-NNN запись
`backward-asks-stakeholder`	BackwardFinding	Stakeholder	who answers
`resolved-by-answer`	BackwardFinding	(timestamp + author)	client-answer record

3.3 SPEC graph (parallel axis)

Edge	From	To	Семантика
`constrained-by`	Requirement	Specification	SR constrained by SPEC-* (typed edge)
`implements-spec`	Task	Specification	TR implements SPEC-*
`depends-on`	Specification	Specification	SPEC depends on another SPEC (DAG)
`referenced-by`	Specification	Requirement / Task	inverse (auto-derived)

3.4 Verification и реализация

Edge	From	To	Семантика
`verifies`	TestCase	Requirement / Specification	TC verifies artifact
`verified-by`	Requirement / Specification	TestCase	inverse (auto-derived)
`implements`	Task	Requirement	Task implements requirement
`realises-in`	TestCase	CodeArtifact	TC.automation.location
`linked-defect`	Requirement	Task (defect type)	Bug на этом требовании

3.5 Происхождение

Edge	From	To	Семантика

`from-order`	ADAPT / Requirement	WorkOrder	source.tz-section reference
`delta-from`	WorkOrder	WorkOrder	delta-T3 → base T3
`cited-in`	Requirement / Specification	WorkOrder	inline citation pointer на раздел T3
`decided`	Requirement / Specification	Decision	Decision при декомпозиции
`deadend`	Requirement / Specification	DeadEnd	Failed approach

3.6 Business / governance

Edge	From	To	Семантика
`owned-by`	Requirement	Stakeholder	business-context.stakeholder
`goal`	Requirement	BusinessGoal	business-context.business-goal
`impacts-kpi`	BusinessGoal	KPI	KPI driven by goal
`compliance-with`	Requirement / Specification	Compliance	compliance entry
`risk-mitigates`	Requirement / Specification	RiskItem	mitigates AIR-NN
`risk-introduces`	Requirement / Specification	RiskItem	introduces new risk (warning trigger)

3.7 Cross-project / integration

Edge	From	To	Семантика
`participates-in`	Specification (SPEC-INT)	Specification (SPEC-INT)	SPEC-INT participants — стороны интеграционного контракта
`cross-deps-on`	Task (project A)	Task (project B)	Cross-project dependency
`integrates-via`	Requirement	Specification (SPEC-INT)	Через SPEC-INT contract

3.8 Edge properties

Edges имеют properties (Cypher-style):

```
(SR:Requirement)-[v:verifies {requirement-version: "1.2", confidence: "high"}]->
(TC:TestCase)
(BR:Requirement)-[c:compliance-with {control: "A.5.34", rationale: "PII
protection"}]->(GDPR:Compliance)
(WO:WorkOrder)-[d:delta-from {effective-date: "2026-05-15"}]->(WO-prev:WorkOrder)
(SR:Requirement)-[cb:constrained-by {since-version: "1.0"}]->(SPEC:Specification)
```

4. Cypher-style query examples

4.4 Stale TC (criteria-version drift)

```
MATCH (r:Requirement)<-[:verifies]-(tc:TestCase)
WHERE tc.last-run.requirement-version < r.version
RETURN r.id, tc.id, tc.last-run.requirement-version, r.version
```

4.5 Multi-hop: code → test → SR → BR → KPI

```
MATCH (code:CodeArtifact {path:"src/auth/registration.py"})
      <-[:realises-in]-(tc:TestCase)
      -[:verifies]->(sr:Requirement {type:"SR"})
      -[:parent*1..2]->(br:Requirement {type:"BR"})
      -[:goal]->(g:BusinessGoal)
      -[:impacts-kpi]->(k:KPI)
RETURN code.path, sr.id, br.id, g.name, k.name
```

«Какие KPI зависят от этого файла кода?» — за один query.

4.6 SPEC dependency cycle detection

```
MATCH path=(s1:Specification)-[:depends-on*]->(s1)
RETURN path
```

Returning rows → нарушение DAG invariant (02-schemas.md §9).

4.7 ADAPT с open backward findings

```
MATCH (a:ADAPT {status:"draft"})-[:contains-backward]->(b:BackwardFinding
{status:"open"})
RETURN a.id, count(b) as open_count
ORDER BY open_count DESC
```

> ***Примечание о нумерации.*** *Используются §4.4-§4.7 для сохранения cross-refs из 02-schemas.md §9 на конкретные queries (Stale TC, SPEC cycle, ADAPT open). Дополнительные queries (BR→KPI, SR-affected tasks, PII→GDPR scope) — derived из patterns §4.4-§4.7 + node/edge таблиц §2-§3.*

5. Derivation rules

KG — derived от frontmatter артефактов. «Прямого редактирования графа» не существует.

Node type	Источник в frontmatter		Edge	Источник
`Requirement`	`id`, `type ∈ {BR,SR,TR}`		`parent`	`parent.id` field
`Specification`	`id`, `type ∈ {SPEC-ARCH..SPEC-OPS}`		`verifies`	`verifies[].id` в TC
`ADAPT`	`id`, `type: ADAPT`		`constrained-by`	`constrained-by[]` в SR
`BackwardFinding`	ADAPT body parsing		`depends-on`	`depends-on[]` в SPEC
`TestCase`	`id`, `type: TC`; derived: `last-run.*`, `last_modified` (§7 SQLite schema), `criteria_changed_at` (§6.5)		`implements-spec`	`implements-spec[]` в TR
`WorkOrder`	TZ-NNN frontmatter		`owned-by`	`business-context.stakeholder` → Stakeholder
`Stakeholder` / `BusinessGoal` / `KPI`	deduplicated из `business-context.*` / `business-outcome.kpi-name`		`compliance-with`	`compliance[]` массив
`Compliance`	`compliance.standard` + `compliance.control`		`derived-from-adapt` / `from-order` / `delta-from`	`source.adapt` / `source.tz-section` / `parent-adapt`

Refresh policy. Граф **rebuilt** при изменении в `.req` репозитории / collection (post-commit/post-save hook), import TC last-run от CI бота, создании/обновлении task. Rebuild **incremental** (только затронутые узлы и грани); full rebuild — раз в неделю reconciliation-агентом.

6. Validation queries (reconciliation)

6.1 Orphan approved requirements

```
MATCH (r:Requirement {status:"approved"})
WHERE NOT (r)-[:verified-by]->()
RETURN r.id  // approved без TC — warning
```

6.3 Circular parent chain

```
MATCH path=(r1:Requirement)-[:parent*]->(r1)
RETURN path
```

6.5 Test fitting suspicious (AIR-06 signal)

```
MATCH (tc:TestCase)
WHERE tc.last_modified < tc.criteria_changed_at
  AND tc.last-run.result = "pass"
RETURN tc.id  // criteria недавно меняли, тут же passing — подозрительно
```

> *Свойства узла `TestCase`: `last_modified` — из node-таблицы (см. §7 SQLite schema); `criteria_changed_at` — derived: timestamp последнего change-record с маркером `[test-spec-change]` (01-glossary.md §2.12). Оба заполняются при derivation графа (§5).*

6.6 SR без constrained-by (missing SPEC links)

```
MATCH (sr:Requirement {type:"SR", status:"approved"})
WHERE NOT (sr)-[:constrained-by]->(:Specification)
RETURN sr.id  // SR approved, но не привязан ни к одному SPEC — warning
```

> ***Дополнительные validation queries*** *(broken citations, orphan stakeholders, orphan SPEC) — derived patterns из §6.1 + §6.5 + node/edge таблиц §2-§3.*

---

7. Нативные для носителя реализации

Граф derived от frontmatter всех `.md` в `<project>.req/` + task store. Рекомендуемая реализация для git-носителя — embedded SQLite с таблицами `nodes(id, type, properties JSON, last_modified)` и `edges(from_id, to_id, edge_type, properties JSON)` с индексами по `from_id`, `to_id`, `edge_type`. Альтернатива для больших проектов: embedded graph DB (Kuzu, RedisGraph local).

Документные носители используют native graph queries через design views / Cypher-like языки — separate infrastructure не требуется.

Schema invariants (независимо от носителя): Типы узлов и типы граней — фиксированы (закрытый список). Properties могут эволюционировать (minor schema bump). Удаление типа узла/грани — major schema bump + migration.

---

8. Federated queries (cross-project)

Координация нескольких проектов через KG federation. Пример: «какие cross-team integrations имеют version drift».

```
MATCH (s1:Specification {project:"auth", type:"SPEC-INT"})
      -[:participates-in]->(int:Specification)
      <-[:participates-in]-(s2:Specification {project:"billing"})
WHERE int.contract-version <> s1.implemented-version
RETURN s1.id, s2.id, int.id, int.contract-version, s1.implemented-version
```

Federation — зависимая от носителя операция; реализуется через convention (межносительный query layer) или native multi-tenant graph.

9. Implementation roadmap

Уровень зрелости	Покрытие графа
RENAR-1 / Core	KG опционален; парсинг frontmatter достаточно.
RENAR-2 / Foundation	Базовый граф: Requirement, TestCase, WorkOrder, Task; edges parent, verifies, verified-by, implements. Простые pre-built queries.
RENAR-3 / Team	+ SPEC, ADAPT, BackwardFinding, Stakeholder, BusinessGoal, KPI, Decision. Edges: constrained-by, depends-on, derived-from-adapt, owned-by, goal, impacts-kpi. AI prompts с graph context.
RENAR-4 / Enterprise	Полная схема + reconciliation queries + federation. Visualization в UI.
RENAR-5	Trend analytics, межносительная federation, embedded graph DB для больших репо.

10. Перекрёстные ссылки

- Каноничные termini узлов и граней — 01-glossary.md.
- Формальные frontmatter schemas (источник derivation) — 02-schemas.md.
- AI Risk Register (AIR-10 KG poisoning) — 03-ai-risk-register.md.
- Style guide (validator использует KG для cross-reference checks) — 04-ai-style-guide.md.

Knowledge Graph Schema RENAR 1.0-draft — renar.tech

ISO/IEC 29148 — матрица трассировки

> ***Назначение:*** *проверяемое соответствие заявления соответствия к ISO/IEC/IEEE 29148:2018 (standard/14 §14.4.2). Нормативные определения полей — в standard/06, standard/08, standard/09, 02-schemas.md.*

RENAR упрощает набор обязательных атрибутов 29148 (18 → 7–8 на артефакт) и добавляет TC как полноценный артефакт, ADAPT и SPEC-ось. Таблица ниже — **полная** трассировка для оценщика соответствия и для заполнения `external-claims[]` в манифесте.

1. Классы требований (29148 §5)

ISO/IEC 29148 класс	RENAR артефакт	Нормативный источник
Stakeholder requirement	`BR`	§6.5
System requirement	`SR` (level: system / subsystem / module)	§6.6
Software requirement (implementation unit)	`TR`	§6.7
Interface / design specification	`SPEC-*` (9 типов)	§8
Verification item	`TC`	§9
Requirements validation (client interpretation)	`ADAPT`	§7

2. Атрибуты требования (29148 Table B.1 → RENAR)

#	ISO/IEC 29148 атрибут	RENAR поле / механизм	Обязательность	Примечание
1	Unique ID	`id` (immutable)	обязательно	V1; см. §3.3.1
2	Requirement statement	body (Потребность / Поведение / ...)	обязательно	EARS-шаблон для SR: §6.6.3
3	Rationale	body «Контекст» + `source.adapt-section`	обязательно (BR/SR)	Прослеживаемость к ADAPT
4	Source	`source.adapt`, `source.tz-section`, `source.document-ref`	обязательно	V5 pinning через `document-ref`

5	Fit criterion	body «Критерии успеха» (BR) / Pass-критерии TC	обязательно	Измеримость через 25022/25023: §14.4.4
6	Priority	`priority` (MoSCoW)	обязательно	WSJF — informative в SAFe mapping
7	Owner	`owner` (BR/SR) / `business-context.stakeholder`	обязательно	§6.5.2
8	Status	`status` (enum жизненного цикла)	обязательно	Машины состояний: §10
9	Verification method	`verified-by[]` → `TC` + `tc-type`	обязательно для verify	TC как полноценный артефакт — расширение 29148
10	Parent / child	`parent.id` , auto `children[]`	обязательно (SR/TR)	Иерархия BR→SR→TR
11	Traceability (derived)	`verified-by` , `constrained-by[]` , `implements-spec[]` , KG edges	derived	reference/05 §4
12	Version	нативная для носителя версия + `requirement-version` в TC	обязательно (V5)	§3.3.5
13	Author	V6 author + `ai-provenance`	обязательно (RENAR-4+ AI)	§4.10.1
14	Date created / modified	timestamps записи изменений носителя	derived (V6)	Журнал аудита: §10.13
15	Risk	`compliance[]` , AIR register link	optional / domain	03-ai-risk-register.md
16	Assumption	ADAPT backward findings `type: assumption`	через ADAPT	§7.4.4
17	Dependency	`depends-on[]` (SPEC), `constrained-by[]` (SR)	обязательно где применимо	DAG invariant: 02 §9
18	Approval authority	QG-0 / QG-2 + ADAPT dual signature	обязательно	Замена formal walkthrough: §14.4.2

Не принято из 29148: review meetings и inspection-only verification без доказательной базы TC — см. §14.7.

3. Verification methods (29148 §6.4)

29148 method	RENAR реализация
Test	`TC` с `tc-type: system | acceptance | contract | …`

Demonstration	`tc-type: acceptance` + client sign-off (QG-4 optional)
Inspection	`[test-spec-change]` workflow + состязательный обзор (§9.13)
Analysis	SR с `quality-characteristic` + eval-TC (`tc-type: eval`) для SPEC-AI

4. Процессы жизненного цикла (29148 §6)

29148 process	RENAR глава	Gate
Requirements elicitation	ADAPT backward + T3	QG-0 (ADAPT approve)
Requirements analysis	BR/SR decomposition	QG-0 (BR/SR approve)
Requirements specification	SPEC axis	QG-3 Architecture (optional/required)
Requirements verification	TC + QG-2	QG-2 Verification
Requirements validation	ADAPT client signature + QG-4	QG-4 Acceptance (optional)
Requirements management	жизненный цикл §10 + носитель V1–V6	Continuous

5. Использование при оценке соответствия

1. Для каждого заявления `ISO/IEC/IEEE 29148:2018` в манифесте — пройти строки §2–§5.
2. Выборочно (≥10% артефактов или все BR/SR уровня system) проверить наличие обязательных полей и трассировку `source.adapt` .
3. Несоответствие любой **обязательной** строки §14 — частичное заявление недопустимо (§14.6.2).

Reference RENAR 1.0-draft — renar.tech

Самооценка соответствия

> ***Назначение:*** *практический kit для Tech Lead / Архитектор перед выпуском RENAR-CONFORMANCE.yaml. Нормативная база — standard/13. Этот документ* ***informative****; при расхождении побеждает standard/13.*

1. Взаимное соответствие MVR ↔ обязательные пункты §13.3

MVR (§0.5)	Положение §13.3	Поле `mandatory-clauses-confirmed`
MVR-1 инверсия источника истины	§13.3.1	`sot-inversion: true`
MVR-2 V1–V6	§13.3.2	`substrate-v1-v6: { v1..v6: true }`
MVR-3 ADAPT per T3	§13.3.3	`adapt-per-tz: true`
MVR-4 9 типов SPEC	§13.3.4	`spec-types-closed-list: true`
MVR-5 TC pos/neg	§13.3.5	`tc-pos-neg-pairing: true`
MVR-6 QG — закрытый список	§13.3.6	`quality-gates-closed-list: true`
MVR-7 манифест соответствия	§13.4 (артефакт)	манифест существует + подписан
— (политика закрытых списков)	§13.3.7	`closed-lists-backward-findings: true`

Все семь MVR + §13.3.7 обязательны для **любого** уровня RENAR-1..5.

2. Чек-лист самооценки (обязательные положения)

Отметьте после проверки доказательной базы в носителе:

§13.3.1 Инверсия источника истины

- ☐ Иерархия BR/SR/SPEC/TC — авторитетный источник о поведении
- ☐ Нет SR, восстановленных из кода без обоснования исправления дефекта
- ☐ Drift-hooks / политика обзора блокируют молчаливую адаптацию SR←код

§13.3.2 Возможности V1–V6 (носитель)

- ☐ V1 immutable history — включён
- ☐ V2 atomic change unit — включён
- ☐ V3 diff & review — включён
- ☐ V4 branching / change-set — включён
- ☐ V5 сквозная фиксация версии между носителями — включена (`verifies[].requirement-version`)

- [ ] V6 author + timestamp — включён

§13.3.3 ADAPT

- [ ] Каждое активное T3 имеет approved ADAPT
- [ ] Каждый delta-T3 имеет delta-ADAPT
- [ ] Двойная подпись (Архитектор + Клиент) зафиксирована

§13.3.4 SPEC types

- [ ] Все SPEC ∈ {ARCH, API, DATA, INT, PROC, UI, AI, SEC, OPS}
- [ ] Нет локальных `SPEC-CUSTOM-*`

§13.3.5 TC pos/neg

- [ ] Каждое верифицируемое утверждение имеет pos + neg TC (или исключение негативного инварианта)
- [ ] QG-2 блокирует `verified` при нарушении

§13.3.6 Quality Gates

- [ ] QG-0, QG-1, QG-2 реализованы как `required`
- [ ] QG-3, QG-4 объявлены `required` | `declared` | `absent` в манифесте
- [ ] Нет локальных пользовательских гейтов

§13.3.7 Закрытые списки

- [ ] Backward finding types — только закрытый список §7.4.4
- [ ] Типы декомпозиции SPEC — только закрытый список §8

Правило: хотя бы один не отмечен → манифест **не выпускать** (§13.5.1).

---

3. Чек-лист уровня (выберите целевой RENAR-N)

Минимум для заявления уровня — standard/11 §11.4–12.8. Краткая сводка:

Уровень	Ключевые доп. критерии
RENAR-1	Только обязательные положения; frontmatter минимален
RENAR-2	Канонический frontmatter + обеспечивается соблюдение статусов жизненного цикла
RENAR-3	Полная ось SPEC + hooks на все QG-0..QG-2
RENAR-4	ai-provenance обязателен; состязательный обзор для SPEC-SEC/AI
RENAR-5	Согласие нескольких моделей; непрерывная оценка; согласование KG

- [ ] Выбранный `level` в манифесте **не выше** фактически пройденного чек-листа

- ☐ `declared-stricter` (если есть) документирован отдельно

4. Шаблон манифеста (минимальный)

Сохранить как `RENAR-CONFORMANCE.yaml` в корне носителя требований:

```yaml
manifest-version: 1
manifest-id: "CFM-YYYY-NNN"
renar-version: "1.0-draft"
senar-version: "1.0"
level: RENAR-2
assessment-date: "2026-05-22"
assessment-type: self
next-assessment-due: "2026-08-22"

mandatory-clauses-confirmed:
  sot-inversion: true
  substrate-v1-v6: { v1: true, v2: true, v3: true, v4: true, v5: true, v6: true }
  adapt-per-tz: true
  spec-types-closed-list: true
  tc-pos-neg-pairing: true
  quality-gates-closed-list: true
  closed-lists-backward-findings: true

quality-gates:
  qg-0: required
  qg-1: required
  qg-2: required
  qg-3: declared
  qg-4: absent

external-claims:
  - standard: "ISO/IEC/IEEE 29148:2018"
    scope: "requirements classes, attributes, lifecycle, verification"
    evidence: "reference/07-iso29148-trace-matrix.md"

substrate:
  type: git
  capabilities-verified: "2026-05-22"
  project-req-ref: "<нативный для носителя pointer>"

signed-by:
  name: "<Architect / Tech Lead>"
  role: approver
  signed-at: "2026-05-22T12:00:00Z"
```

Полный список полей — §13.4.2.

5. Периодичность

- Самооценка: **квартально** (по умолчанию)
- После delta-ТЗ с воздействием на обязательные положения — **внепланово**
- Триггеры потери соответствия — §13.8

Reference RENAR 1.0-draft — renar.tech

Профиль реализации для агента

> ***Назначение:*** *informative contract для агента или нативного для носителя runtime, который* ***имплементирует*** *RENAR без догадок. Нормативный текст — `standard/`; этот документ — operational mapping без привязки к конкретному vendor tooling.*
> ***Машиночитаемо:*** `normative-index.yaml`

1. Как читать таблицу

Колонка	Смысл
`clause_id`	Стабильный ID из `normative-index.yaml`
`Действие агента (абстрактно)`	Что должен уметь runtime **без** vendor CLI
`Входы / выходы`	Артефакты носителя
`Gate`	Контрольная точка качества или проверка под управлением runner

2. MVR ↔ действия агента

clause_id	Действие агента (абстрактно)	Входы	Выходы	Gate
MVR-1	Блокировать reverse-engineering SR/SPEC из кода без bug-fix justification; источник истины = иерархия требований	code diff, SR/SPEC	audit record / blocked promotion	—
MVR-2	Проверить, что носитель декларирует V1–V6 в манифест; прогнать capability checks	`RENAR-CONFORMANCE.yaml`	pass/fail report	—
MVR-3	Реактивный стадийно-независимый ADAPT: создавать ADAPT (0..N на T3) только при findings состязательного обзора; нет findings → source.tz-section + adversarial-review-ref; delta-T3 → тот же реактивный паттерн; дезавуирование через superseded	TZ, состязательный вердикт, ADAPT draft	ADAPT approved / вердикт-свидетельство	QG-ADAPT-approve
MVR-4	Отклонять SPEC с type ∉ закрытого списка	SPEC frontmatter	validation error	QG-spec-approved
MVR-5	Обеспечить pos/neg пару TC на каждое нормативное утверждение	SR/SPEC, TC set	paired TC	QG-2

MVR-6	Отклонять custom gate types; требовать QG-0..2	project config	манифест	—
MVR-7	Требовать подписанный `RENAR-CONFORMANCE.yaml` с `senar-version`	манифест	claim соответствия	—

3. Взаимное соответствие MVR ↔ обязательные положения §13.3

Полное взаимное соответствие MVR ↔ §13.3 — `08-conformance-self-assessment.md §1`. Агент **не должен** считать проект соответствующим, если любое поле `mandatory-clauses-confirmed` = false.

4. Контракт gate runner (абстрактно)

Gate	Pre (проверки агента)	Post (записи агента)
QG-0	Schema valid; parent links; ADAPT exists for TZ-backed SR	`approved` transition + audit-trail
QG-1	TR links `implements-spec[]`; носитель реализации pinned (V5)	TR `in_progress → done` evidence
QG-2	All `verified-by` TC `passing`; pos/neg; `last-run.requirement-version` match	artifact → `verified`

Decision trees: standard/09 §9.1.1, standard/10 §10.1.1.

5. Правило нейтральности к носителю для implementer-ов

Runtime **должен** map-ить abstract actions на нативные для носителя primitives через `RENAR-CONFORMANCE.yaml#v1-v6-mapping` (§3.7). Vendor-specific команды **не** могут быть единственным способом выполнить обязательное положение.

6. Статус покрытия (v1.0-draft)

Область	Index entries	Profile rows	Примечание
MVR	7/7	7/7	complete
§13.3 обязательные	7/7	via bijection	complete
QG-0..2	3/3	3/3	QG-3/4 deferred optional
Обязательные положения по главам	partial	—	expand in later pass

Сопоставление с внешними стандартами

> ***Informative.*** *Детализация standard/14 §14.5. Не изменяет обязательные положения.*

1. SAFe 6.0

Framework масштабированного Agile. RENAR заимствует маппинг иерархии (§4.13.1): Portfolio Epic → BR, Feature → SR, Story → TR. Встроенное качество (TC до approved), WSJF для поля `priority`.

2. Spec-Driven Development

Индустриальный термин 2024–2025: при AI-ускорении критична корректность спецификации. RENAR формализует инверсию источника истины (§2.3.1) — нормативная структура, не привязанная к одному vendor-tool.

3. EARS (Mavin et al., 2009)

Шаблоны контролируемого естественного языка для SR (§6.6.3) и Pass/Fail в TC.

4. BDD / Gherkin / Specification by Example

Prior art для полноценного TC (§9): Given/When/Then, pos/neg как блокирующий gate, version pin V5.

5. NIST AI RMF 1.0

Govern / Map / Measure / Manage — функциональное сопоставление с ролями RENAR (§5), метриками (§12), жизненный цикл deprecate.

6. IEEE 830-1998 (deprecated)

Отозван в пользу ISO/IEC/IEEE 29148. Нормативный правопреемник — §14.4.2.

7. BABOK v3

Gap: elicitation вне scope (T3 уже зафиксировано); Solution Evaluation — частично QG-4 и §12.5.

8. PMBOK 7

«Принципы вместо процессов» — RENAR нормирует **что**, не **как** (§2.5).

9. ISTQB Foundation

Словарь тестирования; `tc-type` совместим с уровнями component/integration/system/acceptance.

10. CMMI v2.0

Prior art для уровней RENAR-1..5 (§11); process-heavy артефакты CMMI не базовый уровень.

11. ISO/IEC 42001:2023 (AIMS)

Организационный governance; RENAR даёт доказательную базу для requirements-среза (ai-provenance, манифест).

12. ISO/IEC 25059:2023

Расширение SQuaRE для AI; vocabulary для `quality-characteristic` AI-компонент.

13. EU AI Act (Reg. 2024/1689)

Поле `ai-act.risk-class` в BR; юридическое соответствие — вне RENAR.

14. SysML / MBSE

Prior art «требования как граф» — reference/05; RENAR выводит граф из текстовых артефактов.

Reference RENAR 1.0-draft — renar.tech