RENAR

Reference appendices (glossary, schemas, registers)

RENAR · Reference · Version 1.0 | 17.07.2026

Authors: Vadim Soglaev, Andrey Yumashev

RENAR Glossary

> ***Purpose:*** *the single source of canonical RENAR terms with examples and a mapping to industry standards. On a wording conflict,* `standard/04-terms.md` *wins* ***normatively****; this glossary is an informative lookup for the reader and the assessor.*

1. Authority chain

When a term is disputed, the following order applies:

1. `standard/04-terms.md` — the normative canon: a standard chapter's definitions win on any conflict.
2. **This document** — informative clarifications and mapping to industry standards; does not override `standard/04`.
3. **ISO/IEC/IEEE 29148** — the international requirements-engineering standard.
4. **BABOK Guide v3** — the *Business Analysis Body of Knowledge*.
5. **A change via an amendment proposal to the full RENAR Standard** — when all sources are silent.

Closed lists: the master index of the closed lists is `standard/01 §1.7.5`.

Not used as a source of terms: bug-tracker tickets, team chats (slang), outdated slide decks, marketing materials.

2. Canonical terms

2.1 Requirement levels

The v1.0 canon is a closed list (see `standard/04-terms.md §4.3`):

RENAR (canonical)	Full name	RU UI label (reference)	Description
BR	Business Requirement	Бизнес-требование	A customer-level business goal: what the organization wants to obtain.
SR	System Requirement	Системное требование	An engineering requirement for the software; verifiable. One `SR` is one verifiable unit.
TR	Task Requirement	Требование к задаче	An implementation-level requirement derived from an `SR`; the unit of work planning.
TC	Test Case	Контрольный пример (TC)	A verifiable artifact that confirms an `SR` / `BR`. Describes behavior, not implementation.

Identification rule: in `frontmatter`, `id` is the canonical RENAR identifier (`BR-01`, `SR-05`). On export to another substrate, a target mapping applies.

Refinements of `SR` (frontmatter fields, not separate artifact types):

- **Module SR** — an `SR` with `level: module` (`standard/06 §6.7`). Formerly the separate label `TM` (§2.1.1).
- **Integration SR** — an `SR` with `constrained-by: [SPEC-INT-N]`. Formerly the separate label `INT-SR` (§2.1.1).
- **Contract TC** — a `TC` with `tc-type: contract`. Formerly the separate label `INT-TC` (§2.1.1).

The `SPEC` family (UX / AI / architecture / integration specifications) — see §2.5 The `SPEC` family.

2.1.1 Legacy labels (deprecated)

When migrating from pre-v1.0 material, deprecated labels appear. Their replacements in the v1.0 canon (`standard/04-terms.md §4.14.1`):

Legacy label	v1.0 canonical replacement	Note
`TM` (Module/Submodule SR)	`SR` with `level: module`	A refinement of `SR`, not a separate artifact type
`UIC` (UI Concept)	`SPEC-UI` (`standard/08 §8.5.6`)	Part of the `SPEC` family (§2.5)
`AIC` (AI Concept)	`SPEC-AI` (`standard/08 §8.5.7`)	Part of the `SPEC` family
`INT-SR` (Integration SR)	`SR` with `constrained-by: [SPEC-INT-N]`	A subclass of `SR`
`INT-TC` (Integration TC)	`TC` with `tc-type: contract`	A subclass of `TC`
`TS` (Technical Specification)	`SPEC-ARCH` or `SPEC-OPS`, depending on content	Part of the `SPEC` family

Migrating existing artifacts: the substrate-native binding to a change set MUST automatically recognize deprecated labels and offer the canonical replacements. The canonical legacy → v1.0 mapping table is `standard/04-terms.md §4.14.1`.

2.2 ADAPT artifact family

Term	Description
ADAPT	The Adaptive Document for Articulating a Project's TZ. An **internal** artifact of interpretation: the forward direction (construal) and the backward direction (findings). Signed by the **Architect only** (`standard/07 §7.5`) — the client never sees it.
ACTZ	The TZ Clarification Protocol (*Agreed Clarification of TZ*). A **contractual** artifact: decisions put to the client and signed by **both parties**. Its contractual name is "TZ Clarification Protocol No. N". Lifecycle: `draft → sent → signed → superseded` (`standard/07 §7.13`). The boundary with ADAPT is drawn **by audience**: shown to the client and approved → an obligation; not shown → an interpretation.

effective TZ	The baseline for delivery and acceptance: the initial TZ (with its annexes) **plus every signed ACTZ**; on a divergence, the later signed document prevails. AT are derived from the effective TZ (`standard/07 §7.14`). ADAPT is **not** part of the acceptance baseline.
`decided-in`	A field of a backward-finding record: a reference to the clause of a **signed** ACTZ in which the decision is recorded. A finding cannot be `resolved` without it (`standard/07 §7.4.5`).
delta-ADAPT	An `ADAPT` for a delta-TZ. Chain: ADAPT-001 → ADAPT-001-delta-1 → ADAPT-001-delta-2; applied in order.
errata-ADAPT	A correction to an approved `ADAPT` (our own interpretation error). It does not alter the frozen document — a separate artifact is added.
Forward (in ADAPT)	The engineering interpretation of each TZ section: quote → interpretation → elaborated scenarios → coverage.
Backward (in ADAPT)	The list of problems and questions for the client. Lifecycle: `open` → `asked-to-client` → `answered` → `resolved` → `frozen` .
Term mapping (in ADAPT)	A "client term → engineering understanding" table.

2.3 Backward categories (closed list)

Every ADAPT backward entry belongs to one of 7 categories. The list is closed; new ones are added only through a change to the full RENAR Standard:

Category	Description
`contradiction`	A contradiction within the TZ.
`gap`	A gap — the TZ is silent on this.
`hidden-assumption`	A hidden engineering assumption that needs confirmation.
`feasibility`	A technically infeasible or expensive requirement.
`regulatory`	Touches legislation / compliance.
`terminology`	An unclear or conflicting client term.
`scope`	A scope-boundary clarification (in / out).

2.4 Quality Gates (closed list, v1.0 canon)

The closed list of RENAR Quality Gates (per `standard/10-lifecycle-qg.md §10.3–10.4`):

Gate	Applies to	Pass condition
QG-0 Approval Gate	`BR` / `SR` / `SPEC` transition: `draft` → `approved`	Goal, acceptance criteria, at least one negative scenario, and coverage; for `SR` — the parent `BR` in `approved` +; for an `SR` with `constrained-by` — the `SPEC` in `approved` +.

QG-1 Implementation Gate	`TC` transition: `draft` → `ready` (**only** for `TC`)	The `TC`'s `verifies` field points to an artifact in `approved`+; `requirement-version` matches; implementation coverage and the version pin are recorded.
QG-2 Verification Gate	`SR` / `BR` transition: `approved` → `verified`	All `TC`s from `verified-by` are green; at least one `TC` with `negative: true`; `last-run` has a `requirement-version` matching the current `version`; the spot-check passed.
QG-3 Architecture Gate (*optional*)	`SPEC-ARCH` approval / the Architect's signature on an ADAPT	An ADR-style artifact is recorded in the substrate; the Architect's signature. Not REQUIRED for declared RENAR conformance.
QG-4 Acceptance Gate (*optional*)	`BR` transition: `verified` → `accepted`	The BR is in `verified`; the business outcome is measured and has reached the threshold; the Stakeholder's signature. Not REQUIRED for declared RENAR conformance.

RENAR conformance: `QG-0` / `QG-1` / `QG-2` are mandatory for declared conformance (`standard/13 §13.3`). `QG-3` / `QG-4` are optional extensions.

The list is closed; new gate numbers are added only through the formal change procedure of the full RENAR Standard.

2.4.1 Legacy QG names (deprecated)

Before v1.0, RENAR used different gate names. The mapping per `standard/04-terms.md §4.14.1`:

Legacy (pre-v1.0)	v1.0 canonical replacement	Note
`QG-0 Context Gate`	`QG-0 Approval Gate`	Rename only; meaning preserved
`QG-1 Requirements Gate`	`QG-1 Implementation Gate`	**Meaning shift:** formerly `BR` / `SR` approval, now only the `TC` transition `draft` → `ready`
`QG-2 Implementation Gate`	`QG-1 Implementation Gate`	Renumbered and merged into a single `QG-1`
`QG-3 Verification Gate`	`QG-2 Verification Gate`	Renumbered
`QG-4 Acceptance Gate`	`QG-4 Acceptance Gate`	Same name; now **optional** for declared RENAR conformance

Migrating existing artifacts: automation rewrites references per the table above. The canonical legacy QG → v1.0 mapping table is `standard/04-terms.md §4.14.1`.

2.4.2 Local ADAPT gates

The `ADAPT` lifecycle (`standard/07-adapt.md`) uses local `ADAPT` gates alongside the canonical `QG-N`. These are not a separate `QG` numbering but local lifecycle events of the `ADAPT`

artifact:

Gate	Application	Pass condition
QG-ADAPT-draft	`ADAPT` creation	The Forward section covers every TZ section.
QG-ADAPT-review	Transition to `review`	All backward entries are `open` or `asked-to-client`; none are `draft`.
QG-ADAPT-asked	Putting the questions to the client	All backward entries are `asked-to-client`; the questions are put to the client as part of an ACTZ (`status: sent`).
QG-ADAPT-answered	After the client answers	All backward entries are `answered`.
QG-ADAPT-approve	`ADAPT` approval	All backward entries are `resolved` with `decided-in` on a clause of a signed ACTZ; the Architect's signature.
QG-ADAPT-frozen	After approval	Immutability; generation of `BR` / `SR` / `SPEC` is permitted.

Local `ADAPT` gates are **not** part of the `QG-0` / `QG-1` / `QG-2` set for declared RENAR conformance. An implementation MAY express `QG-ADAPT-approve` as a local alias for `QG-3` (`Architecture Gate`, where applicable) or store it separately — at the substrate's discretion.

2.5 The SPEC family (closed list)

Type	Purpose	Source
SPEC-ARCH	System / subsystem architecture: contexts, containers, components, deployment view, quality attributes	ADAPT Forward section
SPEC-API	API contracts (REST / GraphQL / gRPC / async events); versioning, error model, rate limits	ADAPT Forward section
SPEC-DATA	Data model: schema, ER diagram, indexes, migrations, storage, personal-data classification	ADAPT Forward section
SPEC-INT	Integration: interaction between subsystems and external systems; protocols, contracts, SLAs	ADAPT Forward section
SPEC-PROC	Process / workflow: business processes, state machines, saga patterns, orchestration and choreography	ADAPT Forward section
SPEC-UI	UI / UX: screens, navigation, user scenarios, accessibility, localization, baseline images	ADAPT Forward section
SPEC-AI	AI / ML: model cards, RAG, prompt engineering, evaluation strategy, cost budget	ADAPT Forward section
SPEC-SEC	Security: authentication / authorization, threat model, secrets management, data classification	ADAPT Forward section, backward regulatory findings

SPEC-OPS	Operations: deployment, observability, SLO / SLA, runbooks, disaster recovery	ADAPT Forward section
SPEC-TEST	Test benches and data: topology, emulators and sandbox instances of external systems, run configuration, datasets on both sides of every integration, reconciliation rules, volume and anonymization; a client signature when real data is used	ADAPT Forward section, integration requirements
SPEC-DOC	Delivered documentation: composition (user manual, administrator manual, training materials), the mandatory sections of each document, version binding, acceptance criteria	ADAPT Forward section, the contractual composition of the delivery

The list is closed — exactly eleven types (canon per `standard/08 §8.3`); new `SPEC` types are added only through a change to the full RENAR Standard.

Three subjects of description are kept apart and MUST NOT be conflated: the nine types (including `SPEC-OPS`) state **how the system is built and how it lives**; `SPEC-TEST` — **how its conformance is proven** (benches, data, run configuration); `SPEC-DOC` — **what enters the delivery** to the client. Hence the boundaries: an administrator manual is always `SPEC-DOC` (the client receives it as part of the delivery), while the vendor team's internal runbook is always `SPEC-OPS`. Every dynamic `TC` is bound to a bench through `environment-ref` pointing at a `SPEC-TEST` (static checks require no bench), and every `SPEC-DOC` is verified by the doc-lint.

2.6 Lifecycle statuses

Status	Applies to	Meaning
`draft`	all artifacts	In progress, not for use by others
`review`	all	Under review, changes possible
`approved`	ADAPT, BR, SR, SPEC	Approved; immutable after signature (for `ADAPT` — the Architect's, §7.5)
`verified`	BR, SR, SPEC	Confirmed by passing `TC` s and the spot-check
`frozen`	ADAPT	Approved and immutable; used as the source for generating `BR` / `SR` / `SPEC`
`deprecated`	all	Outdated, not used in new artifacts; the replacement (if any) is given by the `replaced-by` field
`obsolete`	TR, SPEC, TC	Terminal status: the artifact is no longer current and is not deleted (V1); `ADAPT` has no `obsolete` state
`superseded`	ADAPT, ACTZ, AR	The terminal supersession status: a previously correct decision is revoked by a later artifact; the record is retained for audit (`standard/07 §7.6.4`)

2.7 Substrate capabilities (V1–V6)

RENAR is not tied to a specific artifact substrate. An artifact MAY reside in any substrate that satisfies the following capabilities.

Normative definition — `standard/03 §3.3`:

Capability	Description
V1 — immutable history	Any past state of an artifact can be addressably restored without loss (the revision chain is preserved for audit).
V2 — atomic change unit	A change to an artifact (or a consistent group) commits as a single transaction: fully succeeds or fully rolls back; intermediate states are not externally visible.
V3 — diff & review	A proposed change can be presented as a diff against a base version and accepted or rejected before it reaches the approved state (the basis of approval and the `QG` gates).
V4 — branching & change sets	Work in progress is separated from the approved Source of Truth; several independent changes proceed in parallel without affecting the Source of Truth.
V5 — cross-substrate version pin	A specific version of an artifact in another substrate can be pinned as a resolvable identifier (the basis of the `verifies[].requirement-version` field).
V6 — author + timestamp	For each atomic edit, an unambiguous author and timestamp are recorded (the basis of the `ADAPT` signature and the `ai-provenance` block).

Example substrates: a distributed VCS (`git` with merge-request review), a document-oriented store with a revision chain and signatures, any DBMS with change history and signatures. RENAR does not require `git` — only the **V1–V6** capabilities.

2.8 AI provenance (`ai-provenance`, canonical fields)

In the `frontmatter` of any AI-generated artifact:

Field	Type	Description
`ai-provenance.generated-by`	string	The model, as `<vendor>-<model>-<version>@<date>`. Example: `anthropic-claude-opus-4-7@2026-05-15`.
`ai-provenance.prompt-template`	string	Path to the prompt template and its version. Example: `prompts/adapt-from-tz.md@v2.1`.
`ai-provenance.context-tokens`	integer	Token count of the input context.
`ai-provenance.output-tokens`	integer	Token count of the model output.
`ai-provenance.generation-time-ms`	integer	Generation time in milliseconds.
`ai-provenance.human-edits`	boolean	`true` if a human edited the text after generation. **For approved artifacts this field MUST be `true`.**

2.9 Test-case types

A closed list — six types (canon per `standard/09 §9.5`):

TC type (`tc-type` field)	ISTQB correspondence	Application
`business`	Acceptance Testing	Verifies that the business goal of a BR is achieved (the internal contour). The former name `acceptance` is withdrawn: there are two acceptances, and one name for two subjects caused confusion.
`ux`	usability extension	Verifies a `SPEC-UI` via a VLM judge model / visual comparison against a baseline.
`system`	System Testing	Verifies SR, SPEC-PROC, SPEC-ARCH.
`contract`	Component Integration Testing	Verifies SPEC-API / SPEC-INT / SPEC-DATA via contract testing (Pact and similar).
`eval`	AI-specific	Verifies a `SPEC-AI` via an evaluator LLM and metrics (BLEU, accuracy, hallucination rate).
`security`	security extension	Verifies a `SPEC-SEC` : security invariants (STRIDE); normatively negative scenarios only (`standard/09 §9.6.4`).

2.9bis AT and test-evidence principles

Term	Description
AT (*Acceptance Test*)	The acceptance test of the **contractual contour** (`standard/09 §9.19`). It is derived by an **isolated agent** exclusively from the effective TZ — without access to ADAPT, BR/SR/SPEC, TC, or the code. It is regenerated before every round of trials. It verifies conformance to the **contract**, not to the interpretation.
Test authorship isolation (P8)	The test is frozen before the implementation is started; it is written by an agent **other than** the one writing the implementation; an edit to the criteria is made only through `[test-spec-change]` (`standard/09 §9.18.1`).
Red history (P9)	A test **never once observed red is not evidence**. The fixing run before the implementation MUST be red (`standard/09 §9.18.2`). Isolation guarantees that the test was written before the code, but not that it **verifies anything**; red history closes that remainder.
Killed mutant	The compensation for red history in the `implementation-originated` class: such a test is written after the code and is born green, so its working order is proved by a killed mutant (`standard/06 §6.13.3`).
`implementation-originated`	The narrow legal class of requirements originated by the implementation: **only** internal technical details with no client-observable behavior; with provenance, human approval, a TC before the merge, and a counter in the drift metrics (`standard/06 §6.13`).

2.10 Links (frontmatter fields)

Field	Purpose
`parent`	Parent in the hierarchy (BR → SR → TR); a single source.
`children`	Child artifacts (auto-derived).

`source.adapt`	The `ADAPT` the artifact is derived from (`BR` / `SR` / `SPEC`). The field is **conditional**: present when an ADAPT was created; omitted when the adversarial reviewer returned «no findings» (`standard/07 §7.4.1`).
`source.adapt-section`	The `ADAPT` section (Forward §N).
`source.tz-section`	The TZ section; always mandatory (the dual traceability chain).
`source.adversarial-review-ref`	A reference to an `AR` — the adversarial-review record; mandatory whenever `source.adapt` is omitted (`standard/07 §7.4.6`).
`verifies` (in `TC`)	The `SR` / `BR` the `TC` covers, with `requirement-version` .
`verified-by` (in `SR`)	The `TC` s that confirm the `SR` (auto-derived).
`derived-from`	Template and version (if the artifact was created from a template).
`replaces` / `replaced-by`	Replacement when status is `deprecated` .
`supersedes` (in the new one)	Which requirement is being superseded.
`linked-tasks`	Tasks implementing the `SR` (via the runtime environment, not via files).

2.11 File-naming convention (default)

Type	Pattern	Example
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`
Subsystem SR	`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`
TZ	`tz/TZ-YYYY-NNN.md`	`tz/TZ-2026-001.md`
Delta-TZ	`tz/TZ-YYYY-NNN-delta-N.md`	`tz/TZ-2026-001-delta-1.md`

UX baseline	`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`

This is the default convention; substrate-native stores MAY use a different layout, provided identifiers are stable (capability **V1**).

2.12 Change-record markers (informative)

In a substrate where atomic changes carry metadata (a commit message in `git`, a change-record description in a document-oriented store), the following markers are permitted:

Marker	Purpose
`[delta:TZ-YYYY-NNN]`	A change driven by a delta-TZ.
`[test-spec-change]`	A change to a `TC`'s pass/fail criteria (a separate approval).
`[baseline-update]`	An update to a UX baseline or an eval dataset (a separate approval).
`[coverage]`	Automatic regeneration of coverage, test-plan, and requirement summaries (a bot).
`[reconciliation]`	A change by a reconciliation agent.
`[multi-model-disagreement]`	An artifact where AI model outputs disagree — requires manual review.
`[AI]`	A prefix for AI-generated changes.

A substrate-native mechanism MAY express these markers as change-record fields, labels, or tags — the details are not normative.

3. Mapping to standards

3.1 Requirement levels

RENAR v1.0 canonical labels and external standards:

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`)	(subcomponent scope, extension)	(subcomponent scope)	Story (sometimes)	`TM` (legacy)	CT модуля
SR (`constrained-by: SPEC-INT-N`)	Interface requirement	Interface solution	Cross-feature integration	(related)	INT-CT (legacy)
TR	(no direct class; refinement of a system / system-element requirement)	Detailed solution requirement	Story	`TK`	T3
TC	Test Case	Verification	Story acceptance test	`test_case`	TK
TC (`tc-type: contract`)	Interface test	Component Integration Testing	Contract test	(related)	INT-TK (legacy)
SPEC-UI	(between BR/SR — design specification)	Stakeholder requirement (UX fragment)	(design level)	(extension)	UIC (legacy)
SPEC-AI	(REQ extension)	(n/a)	Enabler	(extension)	AIC (legacy)
SPEC-ARCH / SPEC-OPS	Design description	Solution component	Enabler tech spec	(extension)	TC (legacy)

Note: the "document store (example enum)" and "SENAR (RU)" columns contain historical labels (`TM` , `UIC` , `AIC` , `INT-CT` , etc.) for traceability with pre-v1.0 systems. The RENAR canon column holds the v1.0 labels per §2.1.1. On export to a document store or a SENAR substrate, a target mapping applies.

3.2 Quality Gates

A mapping of RENAR v1.0 canonical `QG-N` to external models. Legacy `QG` names (§2.4.1) are retained in the SENAR column for historical traceability.

RENAR (v1.0)	SENAR (legacy mapping)	Document store (example)	CMMI activity
QG-0 Approval Gate	QG-0 (context)	`VK-1` (start)	Requirements review before commitment
QG-1 Implementation Gate	QG-2 (implementation, legacy)	`VK-1`	Implementation baseline (`TC` readiness)

QG-2 Verification Gate	QG-3 (verification, legacy)	`VK-2`	Verification
QG-3 Architecture Gate *(optional)*	(n/a)	`VK-3` (partial)	Architecture-decision approval
QG-4 Acceptance Gate *(optional)*	QG-4 (Acceptance)	`VK-4`	Customer acceptance

Note: before v1.0, `QG-1 Requirements Gate` is effectively split between the canonical `QG-0 Approval Gate` (`BR` / `SR` / `SPEC` approval) and `QG-1 Implementation Gate` (`TC` readiness). See §2.4.1 for the full mapping.

3.3 Lifecycle statuses

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 Process artifacts

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

3.5 User-interface projections

`frontmatter` fields, identifiers, and file names are always canonical (latin). RU labels are permitted in the UI:

Canonical	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

A UI projection does not replace the canonical identifiers in the substrate.

4. Forbidden / deprecated terms

RENAR does not use the following terms (even where they appear in SENAR / industry literature):

Term	Use instead	Why
User Story as a requirement	SR	A story is a unit of planning, not a requirement. A story MAY implement an SR, but is not itself a requirement.
Use Case (formally)	SPEC-UI + SR	A use case mixes UX and behavior. RENAR separates SPEC-UI (the UX baseline) and SR (behavior).
Spec (without a qualifier)	SR / BR / SPEC-API / SPEC-DATA / ...	"Spec" is ambiguous. Use the precise terms.
Business logic as a requirement	SR	"Business logic" is a code term, not a requirements term.
Functionality	`SR` / `TR`	Too broad; not unambiguously verifiable.
Feature (loose use)	Feature (SAFe context) or SR (the canonical RENAR term)	Ambiguous without a frame of reference.
Wish / "nice-to-have"	(never)	A contractual document is not written this way.
Epic as a requirement	BR (business level) or Portfolio Epic (SAFe)	An epic is a unit of planning, not a requirement.

"Test it by hand"	spot-check (Core Rule 5) or a manual TC with `tc-type: business`	Vague; no verifiable evidence.
"To finish later" (as a status)	`draft / review`	Not from the closed lifecycle list.
TODO in `frontmatter`	a backward entry in the `ADAPT` (if about a requirement) or a task in the tracker (if about implementation)	Open questions live in the right artifact, not in a field comment.

On such findings in project-local artifacts, raise a substrate-side warning (`pre-commit` in `git` , a validation rule in the document store).

5. Glossary versioning

The glossary is a standalone document with its own version. A change to a canonical term is a major-version bump (1.0 → 2.0) and a migration scenario for all project artifacts.

Current version: 1.0-reconciled (phase-1.5 reconciliation with `standard/04-terms.md §4.14.1`).

5.1 Open questions — closed (phase 1.5, 2026-05-16)

Four open questions from the earlier draft were closed by reconciliation with `standard/04-terms.md §4.14.1` :

#	Was an open question	Outcome	Source
1	Canonical language: latin (`BR` / `SR`) or Russian (БТ/ СТ)?	**Canon is latin**; Russian only in the UI projection (§3.5)	`standard/04 §4.13.3` + `reference/06-ru-style-guide.md §1.3`
2	`TM` as a separate label or an `SR` refinement?	**`SR` with `level: module`** — a refinement, not a separate artifact type	`standard/04 §4.14.1` (`TM` deprecated) + §2.1.1
3	AIC ← AAC / AIA / AIC?	**`SPEC-AI`** (v1.0 canon); AIC is a deprecated label	`standard/04 §4.14.1` + §2.1.1
4	`INT-TC` a separate type or a naming convention?	**`TC` with `tc-type: contract`** — a refinement, not a separate type	`standard/04 §4.14.1` (`INT-TC` deprecated) + §2.1.1

5.2 Reconciliation history

Date	Version	Change
2026-05-16	1.0-reconciled	Phase-1.5 reconciliation with `standard/04-terms.md §4.14.1` . Deprecated labels moved from the `§2.1` table to `§2.1.1` ; `QG` names aligned with the v1.0 canon in §2.4; deprecated names → `§2.4.1` . Mapping tables §3.1 and §3.2 updated. Four open questions closed (§5.1). Task: `ru-reconcile-glossary-vs-standard` .

(early drafts)	1.0	Draft fill-in during phase 7; had open questions and divergences from `standard/04 §4.14.1`. Commit history recorded; replaced by the 1.0-reconciled release.

5.3 Cross-references

- **EN Style Guide** (`reference/en/06-en-style-guide.md`) — EN editorial rules for normative text; §1.9 fixes the canonical term list alongside this glossary. On a conflict, editorial-pass wording priority belongs to the Style Guide.
- **Canonical definitions** (`standard/04-terms.md`); §4.14.1 — the deprecated → canon mapping.

RENAR Glossary 1.0-reconciled (EN) — part of `reference/`. See also 02-schemas.md, 03-ai-risk-register.md, 06-en-style-guide.md.

Schemas (formal)

> ***Purpose:*** *machine-readable YAML frontmatter schemas for all RENAR artifact types. Used by substrate-native validators to check conformance.* ***Normative structure definitions*** *live in* `standard/06`, `standard/07`, `standard/08`, `standard/09`. *Example validation:* `node scripts/validate-schema-examples.js`. *This document is a reference (informative lookup).*

1. Common frontmatter (all requirement and SPEC artifacts)

Fields common to BR/SR/TR/SPEC-* (canonical v1.0). Legacy types `UIC` / `AIC` / `INT-SR` / `TS` are deprecated in v1.0 (standard/04 §4.14.1).

```
# 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 | SPEC-TEST | SPEC-DOC
slug: "<kebab-case>"

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

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

# Provenance (conditional, standard/07 §7.4.1): ADAPT is reactive.
# Findings present → source.adapt + adapt-section mandatory. No findings →
adversarial-review-ref mandatory.
# source.tz-section — always mandatory.
source:
  adapt: "ADAPT-NNN"                  # conditional
  adapt-section: "Forward §N.N"       # mandatory if adapt present
  tz-section: "§N.N"                  # always mandatory
  adversarial-review-ref: "AR-NNN"    # mandatory when adapt omitted → AR
(standard/07 §7.4.6)
  document-ref: "<link>"              # pinned revision of source document
  implementation-originated: {}       # conditional; see §1.1 (standard/06 §6.13)

# Hierarchy
parent: { id: "<parent-id>", ref: "<link>" }   # id required for SR (→BR), TR
(→SR); optional for BR
```

```
children: []                          # auto-derived

# Link to SPEC (graph)
constrained-by: []                    # SR → SPEC-* (typed edges)
implements-spec: []                   # TR → SPEC-*
depends-on: []                        # between SPEC

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

# AI provenance (RENAR-4+ mandatory for 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 for 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 }

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

1.1 `source: implementation-originated` — the implementation-originated requirement

A narrow legal origin class for BR / SR / SPEC (standard/06 §6.13): an **internal technical detail** added during implementation (a defensive check, internal validation, logging, an edge-case branch). Client-observable behaviour MUST NOT be covered by this class — it passes only through the contractual contour (ADAPT / ACTZ with signatures).

```
source:
  tz-section: "§N.N"                  # always mandatory (here too)
  implementation-originated:
    change-unit-ref: "<link to the implementation change unit>"   # mandatory —
provenance
    rationale: "<why the functionality was deemed necessary>"     # mandatory
    human-approval:                                               # mandatory; V6
      approved-by: "<human supervisor name>"
      role: "<role>"
      approved-at: "<ISO-datetime>"
    observable-by-client: false       # mandatory; true → non-conformance
```

```
(standard/06 §6.13.4)
    covering-tc: "TC-NN"              # mandatory; the TC exists and passes
before the change is merged
    mutation-check:                                                  # mandatory —
compensates for the absent red history
      mutants-killed: integer         # >= 1; otherwise the TC is not evidence
(standard/09 §9.18.2)
      report-ref: "<link to the mutation-check report>"
```

Rule	Level
`human-approval` is mandatory; an agent MUST NOT legalise its own addition	normative
`covering-tc` exists and passes **before** the change is merged	hook-enforced
`covering-tc` MUST NOT have a red history; a killed mutant is required instead (`mutation-check.mutants-killed >= 1`)	hook-enforced
`observable-by-client: true` together with `implementation-originated` is a non-conformance (an obligation to the client MUST NOT arise from code)	normative
The share of artifacts of this class is a counter in the drift metrics (standard/12 §12.3)	normative

2. BR — Business Requirement

```
# Extends common §1, additionally:

level: system | subsystem             # BR at module level is prohibited
(standard/06 §6.4)
scope: { system: "<system-id>", subsystem: "<subsystem-id>" }

# Cross-level link: subsystem BR → system BR (standard/06 §6.8.2)
implements:                             # array; substrate-agnostic reference
  - { id: BR-NN, scope: { system: "<system-id>" }, rationale: "<short>" }   #
rationale optional
implemented-by: []                      # auto-derived (reverse edge; not written
by the author)

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

# business-outcome — required for 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 }
```

Fields `implements[]` / `implemented-by[]` — normative rules

Rule	Level
`implements[]` required when `level: subsystem` AND the parent system has ≥1 approved BR	recommended v1.0; mandatory v1.1+
The target BR MUST be in status `approved` or higher at the moment this BR is approved	hook-enforced
Cycle detection: the `implements` chain MUST NOT form cycles	hook-enforced
`implements[]` is **not** a parent edge; the ban on multiple parents (standard/06 §6.8.3) applies to SR/TR, not to BR	normative
Deprecate target BR → cascade-warning for all `implemented-by` (not cascade-deprecate)	hook-enforced
Cross-substrate syntax: `id + scope.system` is substrate-independent	normative
Cardinality: array (0..N)	normative

The enforcement gate is `scripts/check-implements-edge.js`. The `implemented-by` field is auto-derived; manual entry is prohibited.

3. SR — System Requirement

```
# Extends common §1, additionally:

parent:
  id: "BR-NN"                          # required

# ADAPT source — via the canonical source.adapt / source.adapt-section (§1).
# There is no separate derived-from-adapt field (standard/06 §6.6.2).

constrained-by:                        # typed edges to SPEC-*
  - "SPEC-UI-NN"
  - "SPEC-API-NN"
  - "SPEC-DATA-NN"
  - "SPEC-SEC-NN"

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

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

4. TR — Task Requirement

```
# Extends common §1, additionally:

parent:
  id: "SR-NN"                          # required

implements-spec: []                    # SPEC-* implemented by this task
estimated-effort: "<short statement>" # optional, free-form
```

5. SPEC-* common schema

All 11 SPEC types share a common structure (§1) plus the following SPEC-specific fields:

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

referenced-by: []                      # auto-derived
```

```
depends-on: []                          # SPEC this one depends on
compliance-refs: []                     # ISO / GDPR / Ф3-152 / AI Act / NIST AI
RMF
```

Mandatory body sections: `## Purpose`, `## Scope`, `## <Type-specific sections — see §6>`, `## Link to requirements`, `## Link to other SPEC`, `## Verification`, `## Open questions`.

---

6. SPEC type-specific extensions

Type-specific fields for the 11 SPEC types. Industry references are in `standard/14`. Legacy replacements: `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 — normative)
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, Φ3-152, 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>" }
```

6.3 SPEC-TEST — benches and data

A test bench is not a deployment environment but a construction of proof (`standard/08 §8.3.2` , §8.5.10). Datasets are described **on both sides** of every integration; the expected result often lives outside our system, so reconciliation rules are a mandatory part of the schema.

```
# SPEC-TEST
benches:                              # bench topology: nodes, versions, what is
live / what is emulated
  - name: "<bench-id>"
    nodes: [{ name: "<node>", version: "<version>", mode: live | emulated }]
    notes: "<bench limitations>"

counterparties:                       # one entry per integration (exactly one
per SPEC-INT)
  - integration: SPEC-INT-NN
    representation: real | sandbox | emulator
    endpoint: "<substrate-native pointer>"
    fidelity-rationale: "<why the representation answers like the real system>"

datasets:                             # on both sides of every integration
  - name: "<dataset-id>"
    side: ours | counterparty
    integration: SPEC-INT-NN          # null for a dataset outside an integration
    volume: "<volume>"
    origin: synthetic | anonymized-production | client-provided
    anonymized: boolean
    contains-real-client-data: boolean   # true ⇒ client-signature is REQUIRED
    retention: "<retention period>"

reconciliation-rules:                 # reconciling results against data held by
foreign systems
  - name: "<rule-id>"
    integration: SPEC-INT-NN
    expected-result-source: "<where the expected result lives when it is not in
our system>"
    match-keys: []
    tolerance: "<reconciliation tolerance>"

run-config:
  mocked: [SPEC-INT-NN]               # what is mocked
  live: [SPEC-INT-NN]                 # what runs live
  run-order: []                       # order of the run
  seed-mechanism: "<how the bench state is prepared>"

client-signature:                     # REQUIRED when real / personal client data
is used
  signed-by: "<name>"
```

```
  role: "<role>"
  organization: "<client-org>"
  signed-at: "<ISO-datetime>"
  signature-ref: "<link>"
```

Rule	Level
At least one `datasets[]` entry with `contains-real-client-data: true` (or `anonymized: false`) ⇒ `client-signature` MUST be filled in; otherwise — **fatal**	hook-enforced
`counterparties[]` MUST hold exactly one entry per integration represented on the bench	hook-enforced
`reconciliation-rules[].expected-result-source` MUST be non-empty when the expected result lives outside our system	normative
A version increment of a SPEC-TEST invalidates `verified` on the TCs that reference it through `environment-ref` (standard/10 §10.5.4)	hook-enforced

6.4 SPEC-DOC — delivered documentation

SPEC-DOC describes the composition of the delivered result: the documents the client receives as part of the delivery (`standard/08 §8.5.11`). The boundary with SPEC-OPS is membership in the delivery: an administrator manual is always SPEC-DOC, a runbook is always SPEC-OPS.

```
# SPEC-DOC
deliverables:                          # one entry per document
  - name: "<document-id>"
    kind: user-manual | admin-manual | training-materials | other
    audience: "<reader role>"
    format: pdf | html | markdown | docx | other
    language: "<delivery language>"

required-sections:                     # mandatory sections — a requirement, not
the vendor's discretion
  - deliverable: "<document-id>"
    sections: ["<mandatory section heading>"]

traces-to: []                          # BR / SR / SPEC whose behaviour the
document describes

version-binding:
  rule: matches-system-version | matches-release-tag | manual
  system-version-ref: "<substrate-native link to the system version>"

acceptance-criteria:                   # what the document is accepted against;
checked by the doc-lint
  - criterion: "<checkable assertion>"
    checked-by: TC-NN                  # a TC with automation.kind: static
(standard/09 §9.8)
```

Rule	Level
The doc-lint is mandatory: a SPEC-DOC MUST have at least one TC with `automation.kind: static`; without it the SPEC-DOC is unverifiable and **does not pass QG-2** (standard/09 §9.8)	hook-enforced
`required-sections[]` MUST be non-empty for every `deliverables[]` entry	hook-enforced
`traces-to[]` MUST be non-empty: the document describes behaviour stated in the requirements; the doc-lint checks coverage of roles and scenarios	normative
Substantive conformance of the text to implemented behaviour is OPTIONAL, via a judge agent (P7 / `eval`); no separate mechanism is introduced	normative

7. ADAPT schema

ADAPT is a separate artifact (`standard/07`). It is reactive: it exists only when there are findings from the adversarial review of the TZ (§7.4.1). On a "no findings" verdict, no ADAPT is created; the review outcome is issued as an AR in either case (§7.1 below), and artifacts reference it via `<artifact>.source.adversarial-review-ref`.

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

# Source
source-tz: { id: TZ-YYYY-NNN, signed-date: "<ISO-date>", signed-by-client: "
<name-role>", document-version-ref: "<substrate-native version identifier>" }   #
V5 pin
parent-adapt: { id: ADAPT-NNN, delta-tz: TZ-YYYY-NNN }   # for delta-ADAPT

# Supersession (standard/07 §7.6.4) — only for superseding-ADAPT
supersedes: ADAPT-MMM                                   # reference to the
superseded ADAPT
superseded-by: ADAPT-NNN                                # auto-derived; on the
superseded one
supersession-rationale: "<contradicting BR/SR/SPEC + source>"   # mandatory if
supersedes present

# Lifecycle (subset of §1: ADAPT does not use verified/deprecated/obsolete;
superseded — terminal on supersession, §7.6.4)
# The client-ready state is REMOVED: putting questions to the client is a matter
for the ACTZ (§7.13), not an ADAPT state (standard/10 §10.8.1).
status: draft | review | asked | answered | approved | frozen | superseded
created: "<ISO-date>"
last-updated: "<ISO-date>"

# Approval (required for approved)
```

```
approval:
  # No client-signature: an ADAPT is an internal interpretation (standard/07
§7.5).
  # The client's obligations are carried by an ACTZ (§7.13) with the dual
signature.
  architect-signature: { signed-by: "<name>", role: architect, signed-at: "<ISO-
datetime>" }

# Auto-derived
generates-requirements: []
generates-specs: []
open-questions-count: integer         # MUST be 0 for 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 entries inside the body:

```
id: B-NNN
category: contradiction | gap | hidden-assumption | feasibility | regulatory |
terminology | scope
status: open | asked-to-client | answered | resolved | revised | 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: "..."                     # how the answer was integrated into
Forward
decided-in: "ACTZ-NNN §M"             # mandatory for status=resolved; a clause
of a signed ACTZ (§7.2)
```

A finding moves to `resolved` only once the decision on it has been **put to the client and signed**: `decided-in` points at clause `§M` of an ACTZ in status `signed` (standard/07 §7.13.2). Referencing an ACTZ in status `draft` is forbidden (standard/07 §7.13.4).

7.1 AR schema — the adversarial-review record

AR (`standard/07 §7.4.6`) is an evidence record capturing the fact and the outcome of the mandatory adversarial review. It is issued in **both** outcomes; on `no-findings` it is itself the evidence referenced by `<artifact>.source.adversarial-review-ref`. It is not a requirements artifact and belongs to no closed list.

```
# Identity
id: AR-NNN                              # sequential within the parent TZ;
immutable
type: AR
tz-ref: TZ-YYYY-NNN                     # the (delta-)TZ under review
trigger-stage: import-tz | decompose-br | decompose-sr | spec | tc   # aligned
with ADAPT.trigger-stage

# Reviewer (isolation: model != primary agent's model, standard/07 §7.10.2)
reviewer: { vendor: "<provider>", model: "<model-id>" }

# Verdict
verdict: findings-present | no-findings
produces-adapt: [ADAPT-NNN]             # mandatory non-empty when findings-
present; empty when no-findings

# Lifecycle
status: draft | issued | superseded
superseded-by: AR-NNN                   # mandatory when status=superseded

# Signature (V6; mandatory for issued)
signature: { author: "<reviewer-id>", timestamp: "<ISO-8601>" }
```

7.2 ACTZ schema — the TZ clarification protocol

ACTZ (`standard/07 §7.13`) is an artifact of the **contractual contour**: a batch of questions, proposals and decisions put to the client and signed by both parties. The boundary with ADAPT is drawn by audience: what is shown to the client and approved is an obligation; what is not shown is interpretation. Cardinality: TZ : ACTZ = 1 : 0..N; ADAPT : ACTZ = 1 : 1..N.

```
# Identity
id: ACTZ-NNN                            # sequential within the parent TZ;
immutable
title: "TZ Clarification Protocol No. N"
type: ACTZ
tz-ref: TZ-YYYY-NNN                     # mandatory; the TZ the protocol belongs to

# Findings closed (conditional)
resolves:                               # B-NNN entries in an ADAPT that the
protocol closes
  - { id: B-NNN, adapt: ADAPT-NNN }     # absent on a client-initiated ACTZ
(standard/07 §7.13.2)

# Decisions (mandatory, non-empty)
decisions:
  - number: "§M"                        # stable clause number; the target of
decided-in
    statement: "<decision in the language of obligations: 'the button is named
X'>"
    tz-section: "§N.N"                  # the TZ section being clarified
```

```
# TZ annexes (conditional; standard/07 §7.13.5)
annexes:
  - { name: "<annex>", version: "<new version>", document-ref: "<link>" }

# Lifecycle
status: draft | sent | signed | superseded
superseded-by: ACTZ-NNN              # mandatory if status=superseded

# Signatures (mandatory for signed; V6)
client-signature: { signed-by: "<name>", role: "<role>", organization: "<client-
org>", signed-at: "<ISO-datetime>", signature-ref: "<link>" }
vendor-signature: { signed-by: "<name>", role: "<role>", signed-at: "<ISO-
datetime>" }
```

Rule	Level
`decisions[].number` is stable and immutable: `B-NNN.decided-in` and `AT.verifies[]` point at it	normative
A signed ACTZ is immutable; a correction is made only by a new ACTZ carrying `superseded-by` on the previous one	hook-enforced
Referencing an ACTZ in status `draft` from `decided-in` is forbidden	hook-enforced
`resolves[]` is absent on a client-initiated protocol; once signed, the decision MUST be reflected in the ADAPT	normative
A TZ annex is not addressable on its own: its new version is approved through `annexes[]` under the same two-sided signature	normative
Effective TZ = the initial TZ (with annexes) + all signed ACTZ; the later signed document prevails (standard/07 §7.14)	normative

8. TC — Test Case

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

# Classification
tc-type: business | ux | system | contract | eval | security   # business — the
former acceptance (standard/09 §9.5)
negative: boolean                    # true for the paired negative 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: "<link>", requirement-version: "<version-ref>"
}   # V5 pinning (standard/03 §3.3.5)

# Pair link (mandatory if negative=false and a pair exists)
paired-with: ["<TC-id>"]

# Task binding (optional; standard/09 §9.19.7)
verifies-tr: "TR-NN"                  # the task to whose scope the test is
narrowed
verifies-claims: []                   # the subset of the parent SR's claims
covered within that TR

# Bench and data (mandatory; standard/09 §9.3, standard/08 §8.5.10)
environment-ref: "SPEC-TEST-NN"       # conditional: mandatory if
automation.kind: dynamic.
                                      # Does not apply to static checks (the doc-
lint of §9.8, the
                                      # structural TC of §9.8.1): the analyzer
works over artifacts

# Automation
automation:
  status: automated | manual-pending
  kind: dynamic | static                    # mandatory; static — the runner is a
static analyser
  location: "<path-to-implementation>"      # mandatory if automated
  runner: pytest | jest | go-test | playwright | vlm-judge | ragas | pact | other
  manual-pending-until: "<ISO date>"        # mandatory if manual-pending
  manual-pending-reason: "<why>"

# Red history (standard/09 §9.18.2) — the condition for counting a TC as evidence
red-history:
  fixing-run:                               # the fixing run, performed before
implementation
    date: "<ISO timestamp>"
    result: fail                            # MUST be red; pass → a signal to
investigate, not a success
    run-ref: "<link>"
  green-transition:                         # the recorded red → green transition
    date: "<ISO timestamp>"
    run-ref: "<link>"
  inherited-from: "TC-NN"                   # conditional: inheritance for tasks
that do not change behaviour
  not-applicable-reason: implementation-originated   # conditional: this class
only; then a killed mutant is mandatory
  mutation-check: { mutants-killed: integer, report-ref: "<link>" }   # mandatory
when not-applicable-reason is set
```

```
# Execution (mandatory if tc-type=ux | eval; judge.vendor ≠ production model
vendor — see §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: "<link>"
  requirement-version: "<version-ref>"
  judge-report: "<for ux/eval>"

# Replacement / obsolescence
obsolete-pending: boolean             # true on detected delta-TZ invalidation
replaces: "<old-id>"
replaced-by: "<new-id>"
obsoleted-date: "<ISO date>"

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

`tc-type: business` is the canonical name; the former `acceptance` is not used in v1.0. Acceptance against the effective TZ is performed not through TC but through AT (§8.1).

8.1 AT schema — the acceptance test of the contractual contour

AT (`standard/09 §9.19`) is derived **exclusively** from the effective TZ (standard/07 §7.14): the initial TZ with its annexes plus every signed ACTZ (§7.2). AT checks conformance to the **contract**, not to the interpretation, and is created by an isolated agent that MUST NOT be given access to the internal contour (ADAPT, BR / SR / SPEC, TC, code).

```
# Identity
id: AT-NN                             # immutable
title: "<short, descriptive>"
type: AT
negative: boolean                     # mandatory; pos/neg pairing — as for TC
(standard/09 §9.7)

# Verification target (mandatory; closed list of references)
verifies:
  - "TZ §N"                           # a section of the effective TZ
  - "ACTZ-NNN §M"                     # a clause of a signed protocol
# A reference to an internal artifact (BR / SR / SPEC / TC) is fatal (standard/09
§9.19.2)

tz-version: "<effective TZ revision>" # mandatory; the revision the AT was
```

```
derived from

# Provenance of the isolated agent (mandatory)
generator:
  vendor: "<provider>"                # MUST differ from the primary agent's
model (mirroring P7)
  model: "<model-id>"
  internal-contour-access: false      # mandatory; confirmation of no access to
the internal contour
  generated-at: "<ISO-8601>"

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

# The acceptance-trial environment and dataset (mandatory; standard/09 §9.19.3,
§8.5.10).
# NOT filled in by the generator: the isolated agent does not see internal
artifacts —
# the Architect or the runner sets this AFTER generation, leaving isolation
intact.
environment-ref: SPEC-TEST-NN

# Automation (mandatory; execution only by an automated runner)
automation:
  status: automated | manual-pending
  kind: dynamic | static
  location: "<path-to-implementation>"
  runner: "<runner>"

# Last run (runner-managed; bot-only)
last-run:
  date: "<ISO timestamp>"
  result: pass | fail | skipped | n/a
  runner-id: "<runner@version>"
  run-ref: "<link>"
  tz-version: "<effective TZ revision at the time of the run>"
```

The body of an AT MUST contain a section with the **verbatim quotation** of the effective-TZ clause under test (`tz_text`) next to the verification steps (standard/09 §9.19.3).

Rule	Level
`verifies[]` contains only `TZ §N` and `ACTZ-NNN §M`; a reference to an internal artifact is **fatal**	hook-enforced
`generator.internal-contour-access: true` is **fatal**: isolation is the substance of the mechanism, not hygiene	hook-enforced
`generator.model` differs from the primary agent's model	hook-enforced
ATs are regenerated before every trial; `tz-version` MUST match the current revision of the effective TZ, otherwise the trials are blocked	hook-enforced

The product is not submitted for hand-over until every AT is in status `passing` (standard/10 §10.4.3)	normative

9. Validation rules (cross-field)

Rules not expressible in pure JSON Schema; they require a custom validator. The "Formal check" column gives an executable predicate or a reference to a ready-made KG query (reference/05 §5/§6).

Rule	Description	Formal check
ID immutable	When a file changes, the `id` field does not change.	`diff(prev.id, curr.id) == ∅`
`parent` exists	For SR — the parent BR exists and is in status ≥ `approved`.	`status(BR[SR.parent.id]) ≥ approved`; orphans — 05 §6.1
`source.adapt` approved	For BR/SR/SPEC — the ADAPT in `source.adapt` is in status `approved` / `frozen`.	`status(ADAPT[art.source.adapt]) ∈ {approved, frozen}`
`verified-by` consistency	TCs in `verified-by` have `verifies[].id` = this artifact.	`∀ 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` for BR/SR/SPEC (conditional)	Canonical ADAPT source when findings are present; on a "no findings" verdict — `source.adversarial-review-ref` (standard/07 §7.4.1). TR — via 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	For an AI artifact, `ai-act.risk-class` is mandatory.	`art.type == SPEC-AI ⇒ art.ai-act.risk-class ≠ null`
Data residency consistency	RU in `SR.data-classification.data-residency` ⇒ the same in the parent BR.	`'RU' ∈ SR.…data-residency ⇒ 'RU' ∈ BR[SR.parent].…data-residency`
Compliance hierarchy	`SR.compliance ⊆ parent BR.compliance` (or explicit justification).	`SR.compliance ⊆ BR[parent].compliance ∨ exists(extension-justification)`
TC `automated` requires location	`automation.status: automated` ⇒ `automation.location` non-empty.	`tc.automation.status == 'automated' ⇒ tc.automation.location ≠ ''`
Negative TC mandatory	For every normative assertion — a TC with `negative: true`.	`∀ assertion ∈ art: ∃ tc(negative: true)` (standard/09 §9.7)

ADAPT open-questions == 0 for approved	Approval is blocked while there are `open` / `asked-to-client` / `answered` / `revised` backward entries: every finding MUST be `resolved` (standard/07 §7.4.5, standard/10 §10.8.2).	`adapt.status == approved ⇒ count(backward[status ∈ {open, asked-to-client, answered, revised}]) == 0` (05 §4.7)
Supersession correct	`supersedes` ⇒ non-empty `supersession-rationale` and a symmetric `superseded-by` on the target; no dangling `source.adapt` on a `superseded` ADAPT (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	The `depends-on` graph between SPEC is a DAG.	cypher cycle-detection (05 §4.6): rows ≠ ∅ ⇒ violation
A `resolved` finding is decided by a signed ACTZ	A backward finding in status `resolved` MUST carry `decided-in: ACTZ-NNN §M` , and the target ACTZ MUST be in status `signed` (standard/07 §7.13.2).	`b.status == resolved ⇒ b.decided-in ≠ null ∧ status(ACTZ[b.decided-in]) == signed ∧ b.decided-in.§M ∈ ACTZ.decisions[].number`
A `signed` ACTZ is signed by both parties	Both signature fields are filled in; a signed protocol is immutable.	`actz.status == signed ⇒ actz.client-signature ≠ null ∧ actz.vendor-signature ≠ null`
AT references the contract only	`AT.verifies[]` contains only `TZ §N` / `ACTZ-NNN §M` ; a reference to an internal artifact is fatal (standard/09 §9.19.2).	`∀ v ∈ at.verifies: v ~ /^(TZ §
AT is fresh against the effective TZ	`AT.tz-version` = the current revision of the effective TZ; a divergence blocks the trials (standard/09 §9.19.4).	`at.tz-version == effective-tz.version`
Red history is the condition for counting a TC	A TC counts as evidence at QG-2 only with a recorded red → green transition; the exception is the `implementation-originated` class with a killed mutant (standard/09 §9.18.2).	`tc.red-history.green-transition ≠ null ∨ tc.red-history.inherited-from ≠ null ∨ (tc.red-history.not-applicable-reason == implementation-originated ∧ tc.red-history.mutation-check.mutants-killed ≥ 1)`
`implementation-originated` is not client-observable	The class legalises an internal detail; client-observable behaviour MUST NOT pass through it (standard/06 §6.13.2).	`art.source.implementation-originated ≠ null ⇒ observable-by-client == false ∧ human-approval ≠ null ∧ mutation-check.mutants-killed ≥ 1`

A dynamic TC is bound to a bench	A TC with `automation.kind: dynamic` MUST carry an `environment-ref` to an existing SPEC-TEST: "the test passed" only means something against a specific bench and specific data (standard/09 §9.3). A missing field on a dynamic TC is **fatal**. Static checks (the doc-lint, the structural TC) require no bench.	`tc.automation.kind == 'dynamic' ⇒ tc.environment-ref ≠ null ∧ type(SPEC[tc.environment-ref]) == 'SPEC-TEST'`
Real client data is signed off by the client	A SPEC-TEST in which at least one dataset carries real or personal client data MUST carry a `client-signature`; volume and anonymization are the client's decision, not the vendor's (standard/08 §8.5.10). A missing signature is **fatal**.	`∃ d ∈ spec.datasets: d.contains-real-client-data == true ∨ d.anonymized == false ⇒ spec.client-signature ≠ null`
A SPEC-DOC is covered by the doc-lint	A SPEC-DOC MUST have at least one doc-lint TC (`automation.kind: static`); without it the SPEC-DOC is unverifiable and **does not pass QG-2** (standard/09 §9.8).	`spec.type == 'SPEC-DOC' ⇒ ∃ tc ∈ spec.verified-by: tc.automation.kind == 'static'`

10. Substrate isomorphism

Mapping for git (YAML frontmatter) ↔ document-oriented store (JSON document):

Field (canonical)	git (YAML frontmatter)	Document store (JSON doc)
`id`	`id`	`_id = <project>:<doc-type>:<slug>` , field `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`

Substrate-native field names MAY differ, but the **semantics and invariants are preserved** through capabilities V1–V6 (01-glossary.md §2.7).

11. Schema versioning

Every artifact has a `schema-version` field (semver). When the file version and the current schema do not match, the validator proposes a migration.

Change	Bump
New optional field	minor (1.0 → 1.1)
New mandatory field	major (1.0 → 2.0) + migration script
Field removal	major + migration script
Enum change	minor if addition, major if removal
Field rename	major + migration script

Current schemas version: 1.0.

12. JSON Schema fragment example (BR)

Key patterns (the full BR schema — `reference/schemas/br.json`, planned):

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$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}$" } } }
  }
}
```

Equivalent JSON schemas for SR/TR/SPEC-*/TC/ADAPT are in `reference/schemas/` (planned).

Schemas reference RENAR 1.0 — see also 01-glossary.md, `standard/06` - `09` for normative definitions.

AI Risk Register for RENAR Projects

> ***Purpose:*** *a register of AI-specific risks for projects that use RENAR (where AI generates requirements, specifications, and tests). Based on ISO/IEC 23894:2023 (AI Risk Management, Annex A risk sources + Clause 6 process per ISO 31000) and NIST AI RMF 1.0. Normative mitigation hooks — standard/09 §9.13, standard/07 §7.10.*

This does not replace the organization's general security risk register. AI risks are a separate class because of the specifics of generation and the unpredictability of models.

1. Register structure

Each risk has:

```
id: AIR-NN                              # immutable
name: "<short name>"
category: hallucination | injection | drift | bias | sgnl-failure | data-quality
| adversarial | privacy
# enum value — the part before the parenthesis; the qualifier in parentheses is
optional (e.g. "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>"]
```

The list AIR-01..AIR-14 is closed; new risks — only through a change to the full RENAR Standard.

Category ↔ NIST AI RMF trustworthiness characteristics (for verifiability of the claim "based on 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

> ***ISO/IEC 23894:2023 references.*** *The AI-risk categories in 23894:2023 are located in* ***Annex A*** *(risk sources); Clause 6 describes the risk-management process (per ISO 31000). The "Annex A — …" descriptors in the register are risk-source labels; the exact mapping to Annex A items is subject to reconciliation when a formal claim is made.*

2. Register AIR-01..AIR-14

Metadata for all 14 risks (Sev=Severity, Like=Likelihood):

AIR	Name	Category	Sev	Like	ISO 23894 (Annex A)	NIST RMF	Status
01	Hallucination in AI-generated requirements	hallucination	**High**	High	Output reliability	Measure	active → mitigated at mature RENAR levels
02	Prompt injection via client TZ	injection	**High**	Low-Medium	Adversarial inputs	Manage	monitoring
03	Model drift / version change	drift	Medium	High	Model lifecycle	Manage	monitoring
04	Bias in AI requirements generation	bias	Medium	Medium	Fairness	Measure	active
05	Single-model failure (no diversity)	sgnl-failure	Medium	High	Single point of failure	Manage	mitigated with full pipeline
06	Test fitting / greening of tests	sgnl-failure	**High**	Medium	Verification integrity	Measure	mitigated with spot-check
07	Hallucinated citations	hallucination	Medium-High	Medium	Output reliability	Measure	monitoring
08	Adversarial inputs in clients data (runtime)	adversarial	**High**	Low	Adversarial inputs	Manage	out-of-scope (application-level)
09	Privacy leakage via AI logs	privacy	**High**	Medium	Privacy	Govern	active
10	Knowledge graph	data-quality	Medium	Low	Data integrity	Map	monitoring

	poisoning						
11	Reconciliation false-positive overload	sgnl-failure (process)	Low-Medium	Medium	Verification integrity	Manage	monitoring
12	Cost runaway (uncontrolled AI spend)	sgnl-failure (operational)	Medium	Medium	Cost governance	Manage	active
13	Stakeholder does not understand AI-generated requirements	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	monitoring

Descriptions + impact + mitigations — below.

AIR-01. Hallucination in AI-generated requirements

When generating BR/SR/SPEC, an AI agent may "make up" statements that are not in ADAPT or the TZ. Impact: scope creep, dispute at acceptance, features that the client did not require. **Mitigations:** source citation (standard/04 §4.10.2 — every statement references ADAPT §N, and when no ADAPT was created (standard/07 §7.4.1) — directly a TZ section via `source.tz-section` + `source.adversarial-review-ref`); adversarial review (a critic model from a different vendor); Hallucination Rate metric ≤1% at mature RENAR levels.

AIR-02. Prompt injection via client TZ

A malicious client may embed hidden instructions for the AI into the TZ ("ignore previous instructions and ..."). Impact: data leakage, malicious changes to requirements, violation of the security policy. **Mitigations:** sandboxing of the AI agent on import (the model treats the TZ as passive data, stated explicitly in the system prompt); the input gateway checks for known injection patterns; a suspicious pattern → escalation, not auto-process.

AIR-03. Model drift / version change

Anthropic / OpenAI / Google update their models — the same prompt with the same TZ may give a different output six months later. Impact: inconsistency between requirements within a single project; inability to reproduce exactly the generation of an old artifact. **Mitigations:** model versioning in `ai-provenance.generated-by` (exact version + date); eval tests for SPEC-AI are run when the model changes; on regeneration — a diff against the old version and an assessment of the changes.

AIR-04. Bias in AI requirements generation

The model has training-data bias — when generating BR it may ignore stakeholders from particular groups (accessibility users, non-English locales, specific regulatory regimes). Impact: requirements do not cover the full spectrum of users; the product is discriminatory or non-compliant. **Mitigations:** multi-model

agreement for `priority=must` BR (different models — different biases); a stakeholder map is mandatory in BR (explicit enumeration); an adversarial critic with the prompt "check for missing stakeholders / accessibility considerations".

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

If all artifacts are generated by a single model, its errors propagate systematically. It "hallucinates" a particular pattern — all requirements inherit it. Impact: systematic distortion of requirements across the project. **Mitigations:** multi-model for `priority=must` BR; an adversarial critic with a different model; isolation of the judge model from the production model in eval tests (SPEC-AI: `judge-model.vendor ≠ production-model.vendor`, see 02-schemas.md §6.2).

AIR-06. Test fitting / greening of tests

An AI agent has a trivial path to greening a failing test — weaken the Pass criterion. This passes code review because "the test is green." Impact: false confidence; defects pass into production. **Mitigations:** the `[test-spec-change]` marker is mandatory for changing Pass/Fail (a separate approval); spot-check of 5 passing TC once per iteration (standard/09 §9.14); the Test-spec Drift Rate metric (standard/12 §12.3).

AIR-07. Hallucinated citations

An AI agent writes the citation `[TZ-2026-001 §4 line 142]`, but in the real TZ §4 line 142 is about something else. The citation looks like evidence, but the evidence is false. Impact: source citation becomes a fiction; the trace chain breaks under audit. **Mitigations:** a citation validator hook (parses the citation, opens the referenced document, verifies conformance); pre-commit/pre-approval block on an invalid citation.

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

The client sends data (via forms, API) deliberately constructed to manipulate an AI component at runtime (not at the requirements-generation stage). Impact: similar to AIR-02, but in production runtime. **Mitigations:** input sanitization at the API gateway; constrained generation (structured outputs only); rate limiting per user. **Status: out-of-scope** — application-level security; RENAR requires SR-level coverage (SPEC-SEC threat model), but runtime protection is a task of implementation, not of normalizing requirements.

AIR-09. Privacy leakage via AI logs

When generating an artifact, an AI agent has PII in its context (from the TZ or an interview). Generation logs (tool events, audit records) may store this PII. Impact: PII ends up in logs, in `ai-provenance`, in training data (if used). **Mitigations:** PII redaction in prompts before sending to the LLM; `data-classification` tracking; disable training on conversations (Anthropic/OpenAI privacy settings, DPA); TTL on event logs with PII.

AIR-10. Knowledge graph poisoning

If the KG is used as primary search for AI agents, an incorrect edge can "poison" all subsequent AI queries that rely on that graph. Impact: the AI generates requirements based on wrong context, systematically.

Mitigations: the KG is derived from frontmatter, not edited directly (see 05-knowledge-graph-schema.md); CI validation of the graph on every change (no circular dependencies, no orphan approved); a reconciliation agent verifies integrity weekly.

AIR-11. Reconciliation false-positive overload

With overly sensitive rules, a reconciliation agent generates many false-positive findings. The Architect starts ignoring them → real findings drown. Impact: reconciliation loses value, the discipline does not scale. **Mitigations:** tunable thresholds in the project configuration; a Reconciliation Findings/Week metric (if it grows without real issues — re-calibration); the Architect may reject findings with a rationale — feedback for tuning the agent's prompt.

AIR-12. Cost runaway (uncontrolled AI spend)

Without budget tracking, AI generation (especially with multi-model, adversarial, eval) may consume tokens disproportionately to project size. Impact: financial losses; the practice becomes unprofitable. **Mitigations:** an `ai-budget` field in frontmatter (target + actual); an aggregated cost metric at project level; a cap per project; an alarm when approached; a recommendation engine "Sonnet/Haiku for routine work, Opus only for `priority=must` BR".

AIR-13. Stakeholder does not understand AI-generated requirements

The AI generates SR in a technical style; the client / a non-technical stakeholder does not understand it during review → approval becomes a formality. Impact: QG-ADAPT-approve / QG-4 acceptance loses meaning; the dispute rate at acceptance grows. **Mitigations:** the style guide (04-ai-style-guide.md); BR in business language (technologies — in SPEC-*, not in BR); a human-readable summary in every BR/SR — a short section, understandable without a technical background.

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

All prompts are optimized for a specific provider (Anthropic Claude). If the provider changes pricing/availability — migration requires rewriting all prompts. Impact: operational risk, costs, business continuity. **Mitigations:** provider-agnostic prompts where possible (avoid vendor-specific tool syntax); multi-model agreement for `priority=must` is normative at RENAR-5 (standard/11 §11.8.1) — guarantees a second provider in the pipeline; periodic test runs on a backup provider.

---

3. Risk matrix

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

Range values from §2 (`Medium-High` , `Low-Medium`) are placed at their upper bound. Critical and High risks in the top-right quadrant are the mitigation priority.

4. Mitigation matrix

Which mitigations cover which risks (compensating mechanisms: a single mitigation is rarely sufficient; high risks require ≥ 2 independent mechanisms):

Mitigation	Covers risks
Source citation (standard/04 §4.10.2)	AIR-01, AIR-07
Adversarial review (a different model)	AIR-01, AIR-04, AIR-05
Spot-check of passing TC (standard/09 §9.14)	AIR-06
Multi-model for `priority=must`	AIR-04, AIR-05, AIR-14
Judge isolation in SPEC-AI	AIR-05
AI provenance (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

Review cadence. Monthly: AIR-01, AIR-02, AIR-06, AIR-07, AIR-09 (high-impact runtime risks). Quarterly: the rest. On-incident: any risk in which an incident occurred → root cause → mitigation.

Owner. Default — the project Architect. For AI-specific risks — the AI Governance Lead (if one exists in the organization).

Storage. The project's risk register is a separate artifact:

```
<project>.req/
  governance/
    ai-risk-register.md          # snapshot of this reference + project-specific
notes
    review-log.md                # review history with dates and signatures
```

On a substrate that does not support directories — an equivalent namespacing.

Reconciliation agent. A weekly run updates the `status` and `last-reviewed` fields of each AIR. If a status changes (e.g., monitoring → active) — an alert to the Architect.

6. Relationship to the standard

AIR	RENAR Core / Standard
AIR-01, 07	Standard ch.4 §4.10.2 (source citation) + ch.7 §7.4.1 (reactive ADAPT: `source.adapt` or `source.tz-section` + `source.adversarial-review-ref`)
AIR-04, 13	Standard ch.5 (roles) + style guide §04
AIR-05	Standard ch.9 §9.6.2 (judge ≠ production isolation) + SPEC-AI
AIR-06	Standard ch.9 §9.7 (pos/neg pairing) + §9.13 (protection from test fitting) + §9.14 (spot-check) + §9.18 (test authorship isolation and red history) + QG-2
AIR-09	Standard ch.4 §4.10.1 (`ai-provenance`) + SPEC-SEC
AIR-12	Standard ch.12 §12.3.9 (Cost per Approved Requirement) + ch.11 §11.8.1 (cost/latency budget)

7. What the risk register does NOT cover

This register focuses on AI-specific risks of the RENAR process. **It does not replace:** the organization's general security risk register (ISO 27001); the compliance risk register (06-compliance.md); the application-level threat model of a specific project (SPEC-SEC). Regulator-mandated requirements (AI Act high-risk, FZ-152) — separate artifacts; this register is the operational tier.

AI Risk Register RENAR 1.0 — renar.tech

Knowledge Graph Schema

> ***Purpose:*** *the formal knowledge-graph (KG) schema for RENAR projects — nodes, edges, properties, query patterns. The KG is a* ***derived view*** *of RENAR artifacts for semantic queries by AI agents and for reconciliation. Machine-readable edge types — §3; the normative link fields — standard/06 §6.10, standard/08.*

The KG is **not the Source of Truth**. The Source of Truth is the artifacts (ADAPT / BR / SR / SPEC / TC) on the substrate. The graph is derived from frontmatter and is never edited directly. If the graph contradicts the artifacts → rebuild the graph, do not edit the artifacts.

1. Use cases

Without the KG, an AI agent has only a flat search (FTS5/RAG + `parent` / `children` direct hops + keyword grep) — a syntactic context. The graph adds a semantic one:

Query	Without the graph	With the graph
All BR affecting the Sales Cycle KPI	Grep + parse frontmatter	A single Cypher query
When SR-05 changes — which tasks and SPEC are affected	Scan all references	Single graph traversal
Which SPEC-AI require high-risk classification under the AI Act	Manually	One query
Stakeholder map for a project	Several queries	Single subgraph extraction
Cross-project dependencies via SPEC-INT	Federation API calls	Federated graph query
Whether requirement SR-12 has started to degrade	Manual analysis	Trend analytic on node properties
Stale TC (verifies an SR whose version was updated, last-run on the old one)	Manual check	One query

2. Node types (closed list)

Type	Source	Identity
`Requirement`	BR / SR / TR artifacts	`<id>`
`Specification`	SPEC-* artifacts (11 types)	`<spec-id>`
`ADAPT`	ADAPT-NNN artifacts	`<adapt-id>`

`BackwardFinding`	B-NNN entries inside an ADAPT	`<adapt-id>:B-NNN`
`TestCase`	TC artifacts (incl. `tc-type: contract`)	`<tc-id>`
`WorkOrder`	TZ and delta-TZ	`<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 (optional)	`<deadend-id>`
`RiskItem`	AI Risk Register entry	`AIR-NN`
`Compliance`	Compliance standard reference	`<std>:<control>`
`Template`	Requirements library template	`<template-id>`

The list is closed; new node types — only via an amendment to the full RENAR Standard.

3. Edge types (closed list)

3.1 Hierarchy and derivation

Edge	From	To	Semantics
`parent`	Requirement	Requirement	A.parent = B → B is parent of A (BR → SR → TR)
`derived-from-adapt`	Requirement / Specification	ADAPT	Artifact derived from an approved ADAPT section
`derived-from-template`	Requirement / Specification	Template	Template (with a 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	Semantics
`from-tz`	ADAPT	WorkOrder	source-tz pointer
`parent-adapt`	ADAPT	ADAPT	delta-ADAPT → root (`parent-adapt`)
`supersedes`	ADAPT	ADAPT	superseding ADAPT → superseded one; inverse `superseded-by` (standard/07 §7.6.4); the target moves to `superseded`
`contains-backward`	ADAPT	BackwardFinding	B-NNN entry
`backward-asks-stakeholder`	BackwardFinding	Stakeholder	who answers
`decided-in`	BackwardFinding	ACTZ	Pointer to the clause of a signed ACTZ that records the client's decision (standard/07 §7.4.5); a pointer to an unsigned ACTZ is fatal

3.3 SPEC graph (parallel axis)

Edge	From	To	Semantics
`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 and implementation

Edge	From	To	Semantics
`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 on this requirement

3.5 Provenance

Edge	From	To	Semantics
`from-order`	ADAPT / Requirement	WorkOrder	source.tz-section reference
`delta-from`	WorkOrder	WorkOrder	delta-TZ → base TZ
`cited-in`	Requirement / Specification	WorkOrder	inline citation pointer to a TZ section
`decided`	Requirement / Specification	Decision	Decision during decomposition
`deadend`	Requirement / Specification	DeadEnd	Failed approach

3.6 Business / governance

Edge	From	To	Semantics
`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	Semantics
`participates-in`	Specification (SPEC-INT)	Specification (SPEC-INT)	SPEC-INT participants — the parties to the integration contract
`cross-deps-on`	Task (project A)	Task (project B)	Cross-project dependency
`integrates-via`	Requirement	Specification (SPEC-INT)	Via a SPEC-INT contract

3.8 Edge properties

Edges carry properties (Cypher-style):

```
(TC:TestCase)-[v:verifies {requirement-version: "1.2", confidence: "high"}]->
(SR:Requirement)
(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
```

"Which KPI depend on this code file?" — in a single query.

4.6 SPEC dependency cycle detection

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

Returning rows → a violation of the DAG invariant (02-schemas.md §9).

4.7 ADAPT with 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
```

> ***Note on numbering.*** *§4.4-§4.7 are used to preserve the cross-refs from 02-schemas.md §9 to specific queries (Stale TC, SPEC cycle, ADAPT open). Additional queries (BR→KPI, SR-affected tasks, PII→GDPR scope) are derived from the patterns in §4.4-§4.7 plus the node/edge tables in §2-§3.*

5. Derivation rules

The KG is derived from artifact frontmatter. "Direct editing of the graph" does not exist.

Node type	Source in frontmatter		Edge	Source
`Requirement`	`id`, `type ∈ {BR,SR,TR}`		`parent`	`parent.id` field
`Specification`	`id`, `type ∈ {SPEC-ARCH..SPEC-DOC}` (all 11 types, standard/08 §8.3)		`verifies`	`verifies[].id` in a TC
`ADAPT`	`id`, `type: ADAPT`		`constrained-by`	`constrained-by[]` in an SR
`BackwardFinding`	ADAPT body parsing		`depends-on`	`depends-on[]` in a SPEC
`TestCase`	`id`, `type: TC`; derived: `last-run.*`, `last_modified` (§7 SQLite schema), `criteria_changed_at` (§6.5)		`implements-spec`	`implements-spec[]` in a TR
`WorkOrder`	TZ-NNN frontmatter		`owned-by`	`business-context.stakeholder` → Stakeholder
`Stakeholder` / `BusinessGoal` / `KPI`	deduplicated from `business-context.*` / `business-outcome.kpi-name`		`compliance-with`	`compliance[]` array
`Compliance`	`compliance.standard` + `compliance.control`		`derived-from-adapt` / `from-order` / `delta-from`	`source.adapt` / `source.tz-section` / `parent-adapt`

Refresh policy. The graph is **rebuilt** on a change in the `.req` repository / collection (post-commit/post-save hook), on import of TC last-run from the CI bot, and on creation/update of a task. The rebuild is **incremental** (only the affected nodes and edges); a full rebuild — once a week by the reconciliation agent.

6. Validation queries (reconciliation)

6.1 Orphan approved requirements

```
MATCH (r:Requirement {status:"approved"})
WHERE NOT (r)-[:verified-by]->()
RETURN r.id  // approved without a 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 changed recently, yet passing right away — suspicious
```

> *Properties of a `TestCase` node: `last_modified` — from the node table (see §7 SQLite schema); `criteria_changed_at` — derived: the timestamp of the last change-record carrying the `[test-spec-change]` marker (01-glossary.md §2.12). Both are populated during graph derivation (§5).*

6.6 SR without constrained-by (missing SPEC links)

```
MATCH (sr:Requirement {type:"SR", status:"approved"})
WHERE NOT (sr)-[:constrained-by]->(:Specification)
RETURN sr.id  // SR approved but not linked to any SPEC — warning
```

> ***Additional validation queries*** *(broken citations, orphan stakeholders, orphan SPEC) — patterns derived from §6.1 + §6.5 plus the node/edge tables in §2-§3.*

7. Substrate-native implementations

The graph is derived from the frontmatter of all `.md` files in `<project>.req/` plus the task store. The recommended implementation for a git substrate is an embedded SQLite with tables `nodes(id, type, properties JSON, last_modified)` and `edges(from_id, to_id, edge_type, properties JSON)`, indexed on `from_id`, `to_id`, `edge_type`. An alternative for large projects: an embedded graph DB (Kuzu, RedisGraph local).

Document substrates use native graph queries via design views / Cypher-like languages — no separate infrastructure is required.

Schema invariants (substrate-independent): node types and edge types are fixed (closed list). Properties MAY evolve (minor schema bump). Removing a node/edge type — a major schema bump + migration.

8. Federated queries (cross-project)

Coordinating multiple projects via KG federation. Example: "which cross-team integrations have 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 is a substrate-dependent operation; it is implemented via a convention (a cross-substrate query layer) or a native multi-tenant graph.

9. Implementation roadmap

Maturity level	Graph coverage
RENAR-1 / Core	KG optional; parsing frontmatter is sufficient.
RENAR-2 / Foundation	Basic graph: Requirement, TestCase, WorkOrder, Task; edges parent, verifies, verified-by, implements. Simple 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 with graph context.
RENAR-4 / Enterprise	Full schema + reconciliation queries + federation. Visualization in the UI.
RENAR-5	Trend analytics, cross-substrate federation, embedded graph DB for large repos.

10. Cross-references

- Canonical termini for nodes and edges — 01-glossary.md.
- Formal frontmatter schemas (the derivation source) — 02-schemas.md.
- AI Risk Register (AIR-10 KG poisoning) — 03-ai-risk-register.md.
- Style guide (the validator uses the KG for cross-reference checks) — 04-ai-style-guide.md.

Knowledge Graph Schema RENAR 1.0 — renar.tech

ISO/IEC 29148 — Trace Matrix

> ***Purpose:*** *verifiable conformance of a conformance claim to ISO/IEC/IEEE 29148:2018 (standard/14 §14.4.2). Normative field definitions are in standard/06, standard/08, standard/09, 02-schemas.md.*

RENAR simplifies the set of mandatory 29148 attributes (18 → 7–8 per artifact) and adds TC as a first-class artifact, ADAPT, and the SPEC axis. The table below is the **complete** trace for a conformance assessor and for populating `external-claims[]` in the manifest.

1. Requirement classes (29148 §5)

ISO/IEC 29148 class	RENAR artifact	Normative source
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-*` (11 types)	§8
Verification item	`TC`	§9
Requirements validation (agreement with the client)	`ACTZ` + `AT`	§7.13, §9.19
TZ interpretation (internal artifact; an extension of 29148)	`ADAPT`	§7

2. Requirement attributes (29148 Table B.1 → RENAR)

#	ISO/IEC 29148 attribute	RENAR field / mechanism	Mandatoriness	Note
1	Unique ID	`id` (immutable)	mandatory	V1; see §3.3.1
2	Requirement statement	body (Need / Behavior / ...)	mandatory	EARS template for SR: §6.6.3
3	Rationale	body "Context" + `source.adapt-section` (when an ADAPT exists) or `source.adversarial-review-ref`	mandatory (BR/SR)	Traceability to ADAPT, or to the AR under a "no findings" verdict (§7.4.1)

4	Source	`source.tz-section` (always); `source.adapt` **or** `source.adversarial-review-ref` (conditional, §7.4.1); `source.document-ref`	mandatory	V5 pinning via `document-ref`
5	Fit criterion	body "Success criteria" (BR) / Pass criteria of TC	mandatory	Measurability via 25022/25023: §14.4.4
6	Priority	`priority` (MoSCoW)	mandatory	WSJF — informative in the SAFe mapping
7	Owner	`owner` (BR/SR) / `business-context.stakeholder`	mandatory	§6.5.2
8	Status	`status` (lifecycle enum)	mandatory	State machines: §10
9	Verification method	`verified-by[]` → `TC` + `tc-type`	mandatory for verify	TC as a first-class artifact — an extension of 29148
10	Parent / child	`parent.id`, auto `children[]`	mandatory (SR/TR)	Hierarchy BR→SR→TR
11	Traceability (derived)	`verified-by`, `constrained-by[]`, `implements-spec[]`, KG edges	derived	reference/05 §3
12	Version	substrate-native version + `requirement-version` in TC	mandatory (V5)	§3.3.5
13	Author	V6 author + `ai-provenance`	mandatory (RENAR-4+ AI)	§4.10.1
14	Date created / modified	substrate change-record timestamps	derived (V6)	Audit log: §10.13
15	Risk	`compliance[]`, AIR register link	optional / domain	03-ai-risk-register.md
16	Assumption	ADAPT backward findings `type: hidden-assumption`	via ADAPT	§7.4.4
17	Dependency	`depends-on[]` (SPEC), `constrained-by[]` (SR)	mandatory where applicable	DAG invariant: 02 §9
18	Approval authority	QG-0 / QG-2 + ADAPT architect signature + ACTZ dual signature	mandatory	Replacement for the formal walkthrough: §14.4.2

Not adopted from 29148: review meetings and inspection-only verification without TC evidence — see §14.7.

3. Verification methods (29148 §6.4)

29148 method	RENAR implementation
Test	`TC` with `tc-type: system | business | contract | …`
Demonstration	`AT` against the effective TZ (§9.19); `tc-type: business` + QG-4 (optional)
Inspection	`[test-spec-change]` workflow + adversarial review (§9.13)
Analysis	SR with `quality-characteristic` + eval-TC (`tc-type: eval`) for SPEC-AI

4. Lifecycle processes (29148 §6)

29148 process	RENAR chapter	Gate
Requirements elicitation	ADAPT backward + TZ	QG-3 (ADAPT approve, §10.4.1)
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	A signed ACTZ (§7.13) + AT against the effective TZ	AT release gate (§10.4.3) — mandatory and not a Quality Gate; QG-4 (optional) measures the business outcome
Requirements management	lifecycle §10 + substrate V1–V6	Continuous

5. Use in conformance assessment

1. For each `ISO/IEC/IEEE 29148:2018` claim in the manifest — walk through the rows of §2–§5.
2. By spot-check (≥10% of artifacts, or all system-level BR/SR) verify the presence of mandatory fields and the `source.adapt` trace.
3. Non-conformance of any **mandatory** row of §2 — a partial claim is not permitted (§14.6.2).

Reference RENAR 1.0 — renar.tech

Conformance Self-Assessment

> ***Purpose:*** *a practical kit for the Tech Lead / Architect before releasing RENAR-CONFORMANCE.yaml. The normative basis is standard/13. This document is* ***informative****; on conflict, standard/13 wins.*

1. MVR ↔ mandatory clauses §13.3 bijection

MVR (§0.5)	§13.3 clause	`mandatory-clauses-confirmed` field
MVR-1 Source-of-Truth inversion	§13.3.1	`sot-inversion: true`
MVR-2 V1–V6	§13.3.2	`substrate-v1-v6: { v1..v6: true }`
MVR-3 ADAPT per TZ	§13.3.3	`adapt-per-tz: true`
MVR-4 11 SPEC types	§13.3.4	`spec-types-closed-list: true`
MVR-5 TC pos/neg	§13.3.5	`tc-pos-neg-pairing: true`
MVR-6 QG — closed list	§13.3.6	`quality-gates-closed-list: true`
MVR-7 conformance manifest	§13.4 (artifact)	manifest exists + signed
— (closed-list policy)	§13.3.7	`closed-lists-backward-findings: true`

All seven MVR + §13.3.7 are mandatory for **any** RENAR-1..5 level.

2. Self-assessment checklist (mandatory clauses)

Check off after verifying the evidence in the substrate:

§13.3.1 Source-of-Truth inversion

- ☐ The BR/SR/SPEC/TC hierarchy is the authoritative source of behavior
- ☐ No SR reconstructed from code without a defect-fix justification
- ☐ Drift-hooks / review policy block silent SR←code adaptation

§13.3.2 V1–V6 capabilities (substrate)

- ☐ V1 immutable history — enabled
- ☐ V2 atomic change unit — enabled
- ☐ V3 diff & review — enabled
- ☐ V4 branching / change-set — enabled
- ☐ V5 end-to-end version pinning across substrates — enabled (`verifies[].requirement-version`)
- ☐ V6 author + timestamp — enabled

§13.3.3 Reactive ADAPT

- [ ] Every TZ has passed the adversarial review; the outcome is issued as an AR in status `issued`
- [ ] On a "findings present" verdict: the ADAPT is in status `approved` with the Architect's signature
- [ ] On a "no findings" verdict: no ADAPT is created; BR/SR/SPEC carry `source.tz-section` + `source.adversarial-review-ref`
- [ ] Every finding in `resolved` carries `decided-in` on a clause of a signed ACTZ
- [ ] Signed ACTZ carry the dual signature (client + contractor)
- [ ] A client signature on an ADAPT is not declared mandatory (admissible only as `declared-stricter`)
- [ ] delta-TZ: the delta-ADAPT is created on the fact of findings from the delta-TZ review, not automatically
- [ ] Superseded ADAPT are retained in the terminal `superseded`; no dangling `source.adapt` remain

§13.3.4 SPEC types

- [ ] All SPEC ∈ {ARCH, API, DATA, INT, PROC, UI, AI, SEC, OPS, TEST, DOC} — the closed list of 11 types
- [ ] No local `SPEC-CUSTOM-*`
- [ ] Test benches and data are described as `SPEC-TEST`, not as a section of `SPEC-OPS`
- [ ] Delivered documentation is described as `SPEC-DOC`; the internal runbook stays in `SPEC-OPS`
- [ ] Every `SPEC-TEST` in which at least one dataset carries real or personal client data has a `client-signature`

§13.3.5 TC pos/neg

- [ ] Every verifiable assertion has a pos + neg TC (or a negative-invariant exception)
- [ ] QG-2 blocks `verified` on violation
- [ ] Every TC with `automation.kind: dynamic` carries an `environment-ref` to an existing `SPEC-TEST` (a missing field is fatal; static checks require no bench)
- [ ] Every `SPEC-DOC` is covered by the doc-lint (a TC with `automation.kind: static`); without it the SPEC-DOC does not pass QG-2

§13.3.6 Quality Gates

- [ ] QG-0, QG-1, QG-2 implemented as `required`
- [ ] QG-3, QG-4 declared `required` | `declared` | `absent` in the manifest
- [ ] No local custom gates

§13.3.7 Closed lists

- [ ] Backward finding types — closed list §7.4.4 only
- [ ] SPEC decomposition types — closed list §8 only

Rule: if at least one is unchecked → **do not release** the manifest (§13.5.1).

3. Level checklist (choose the target RENAR-N)

The minimum for claiming a level is standard/11 §§11.4–11.8. Brief summary:

Level	Key additional criteria
RENAR-1	Mandatory clauses only; frontmatter minimal
RENAR-2	Basic frontmatter (no strict schema validation); TZ and delta-TZ as explicit immutable artifacts; lifecycle statuses not yet mandatory
RENAR-3	Full frontmatter schema + lifecycle statuses; hooks enforce QG-0 and QG-1, QG-2 — partially
RENAR-4	ai-provenance mandatory; pos/neg pairing for every normative assertion; QG-2 enforced natively
RENAR-5	Adversarial review as a gate; multi-model consensus for `priority: must`; knowledge graph as primary lookup; continuous evaluation of SPEC-AI

- [ ] The chosen `level` in the manifest is **no higher** than the checklist actually passed
- [ ] `declared-stricter` (if present) is documented separately

4. Manifest template (minimal)

Save as `RENAR-CONFORMANCE.yaml` at the root of the requirements substrate:

```yaml
manifest-version: 1
manifest-id: "CFM-YYYY-NNN"
renar-version: "1.0"
senar-version: "1.0"
level: RENAR-2
assessment-date: "2026-05-22"
assessment-mode: self          # self | third-party (§13.4.2)
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
```

```yaml
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"
    clause-ref: "§14.4.2"
    claim: "partial"           # full | partial | aligned

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: "<substrate-native pointer>"

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"]

assessor:
  id: "<V6 author identifier>"
  role: architect              # architect | authorized-role-holder | external-
assessor
  signature-ref: "<substrate-native pointer to the signature event>"
```

The full field list is §13.4.2.

5. Cadence

- Self-assessment: **quarterly** (default)
- After a delta-TZ that affects mandatory clauses — **out of cycle**
- Conformance-loss triggers — §13.8

Reference RENAR 1.0 — renar.tech

Agent Implementation Profile

> ***Purpose:*** *an informative contract for an agent or substrate-native runtime that* ***implements*** *RENAR without guesswork. The normative text is* `standard/` *; this document is an operational mapping with no ties to any specific vendor tooling.*
> ***Machine-readable:*** `normative-index.yaml`

1. How to Read the Table

Column	Meaning
`clause_id`	Stable ID from `normative-index.yaml`
`Agent action (abstract)`	What the runtime MUST be able to do **without** a vendor CLI
`Inputs / outputs`	Substrate artifacts
`Gate`	A quality checkpoint or a check driven by the runner

2. MVR ↔ Agent Actions

clause_id	Agent action (abstract)	Inputs	Outputs	Gate
MVR-1	Block reverse-engineering of SR/SPEC from code without bug-fix justification; the Source of Truth = the requirements hierarchy	code diff, SR/SPEC	audit record / blocked promotion	—
MVR-2	Verify that the substrate declares V1–V6 in the manifest; run capability checks	`RENAR-CONFORMANCE.yaml`	pass/fail report	—
MVR-3	Reactive, stage-independent ADAPT: create an ADAPT (0..N per TZ) only on findings from adversarial review; no findings → source.tz-section + adversarial-review-ref; delta-TZ → the same reactive pattern; supersession via superseded	TZ, adversarial verdict, ADAPT draft	ADAPT approved / verdict evidence	QG-ADAPT-approve
MVR-4	Reject a SPEC whose type ∉ the closed list	SPEC frontmatter	validation error	QG-0
MVR-5	Ensure a pos/neg pair of TCs for every normative statement	SR/SPEC, TC set	paired TC	QG-2

MVR-6	Reject custom gate types; require QG-0..2	project config	manifest	—
MVR-7	Require a signed `RENAR-CONFORMANCE.yaml` with `senar-version`	manifest	conformance claim	—

3. MVR ↔ Mandatory Clauses §13.3 Bijection

The full MVR ↔ §13.3 bijection is in `08-conformance-self-assessment.md §1`. The agent MUST NOT consider a project conformant if any `mandatory-clauses-confirmed` field = false.

4. Gate Runner Contract (abstract)

Gate	Pre (agent checks)	Post (agent records)
QG-0	Schema valid; parent links; source resolved: `source.adapt` to an ADAPT in `approved` **or** `source.tz-section` + `source.adversarial-review-ref` to an AR with status `issued` (§7.4.1)	`approved` transition + audit-trail
QG-1	TC only (§10.3.2): `automation.status: automated` (valid `automation.location`) or `manual-pending`; implementation pinned to the artifact version (V5); pos/neg pairing; mandatory TC body sections (§9.4)	TC `draft` → `ready` + transition-log record
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. Substrate-Neutrality Rule for Implementers

The runtime MUST map abstract actions onto substrate-native primitives via `RENAR-CONFORMANCE.yaml#substrate-capabilities` (§3.7). Vendor-specific commands MUST NOT be the only way to satisfy a mandatory clause.

6. Coverage Status (v1.0)

Area	Index entries	Profile rows	Note
MVR	7/7	7/7	complete
§13.3 mandatory	8/8	MVR-1..MVR-6; §13.3.7 / §13.3.8 are outside any MVR	no bijection: MVR-7 maps to §13.4
QG-0..2	3/3	3/3	QG-3/4 deferred optional

Mandatory clauses by chapter	partial	—	expand in later pass

Mapping to External Standards

> ***Informative.*** *Elaboration of standard/14 §14.5. Does not alter the mandatory clauses.*

1. SAFe 6.0

A scaled-Agile framework. RENAR borrows the hierarchy mapping (§4.13.1): Portfolio Epic → BR, Feature → SR, Story → TR. Built-in quality (TC up to approved), WSJF for the `priority` field.

2. Spec-Driven Development

An industry term from 2024–2025: under AI acceleration, the correctness of the specification becomes critical. RENAR formalizes the Source-of-Truth inversion (§2.3.1) — a normative structure not tied to a single vendor tool.

3. EARS (Mavin et al., 2009)

Controlled-natural-language templates for SR (§6.6.3) and Pass/Fail in TC.

4. BDD / Gherkin / Specification by Example

Prior art for a full-fledged TC (§9): Given/When/Then, pos/neg as a blocking gate, version pin V5.

5. NIST AI RMF 1.0

Govern / Map / Measure / Manage — a functional mapping to RENAR roles (§5), metrics (§12), and the deprecate lifecycle.

6. IEEE 830-1998 (deprecated)

Withdrawn in favor of ISO/IEC/IEEE 29148. The normative successor is §14.4.2.

7. BABOK v3

Gap: elicitation is out of scope (the TZ is already fixed); Solution Evaluation — partially QG-4 and §12.5.

8. PMBOK 7

"Principles over processes" — RENAR standardizes **what**, not **how** (§2.5).

9. ISTQB Foundation

A testing vocabulary; compatible with RENAR terminologically, but there is no value-for-value correspondence: TC types are RENAR's own closed list `tc-type` (§9.5): `business` / `ux` / `system` / `contract` / `eval` / `security`; test levels are expressed by the separate `level` field (§9.3): `system` / `subsystem` / `module`.

10. CMMI v2.0

Prior art for the RENAR-1..5 levels (§11); CMMI's process-heavy artifacts are not the baseline level.

11. ISO/IEC 42001:2023 (AIMS)

Organizational governance; RENAR provides the evidence base for the requirements slice (ai-provenance, manifest).

12. ISO/IEC 25059:2023

An extension of SQuaRE for AI; a vocabulary for the `quality-characteristic` of AI components.

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

The `ai-act.risk-class` field in BR; legal conformance is outside RENAR.

14. SysML / MBSE

Prior art for "requirements as a graph" — reference/05; RENAR derives the graph from textual artifacts.

Reference RENAR 1.0 — renar.tech

Document templates BR / SR / TR / TC / AR / ACTZ / AT

> ***Status:*** *informative. This is a* ***possible implementation*** *of the normative schemas — organizations may adapt the skeletons to their substrate and editorial conventions. The normative text these templates merely illustrate:* `standard/06` *(BR / SR / TR),* `standard/07 §7.4.6` *(AR),* `standard/07 §7.13` *(ACTZ) and* `standard/09` *(TC, AT). Where a skeleton and a chapter disagree,* ***the chapter prevails****.*
>
> ***The closed list is untouched:*** *this appendix provides skeletons for already-normative types; new artifact or SPEC types are added only through the formal standard-change procedure (chapter 13). AR is an evidence record, not a requirement type, and belongs to no closed list (§1.7.5).*

12.1 Status and how to use it

Each section below carries two skeletons: a `yaml` block with the frontmatter (with per-line comments on field obligation) and a `markdown` block with the body skeleton. To get a ready artifact file, concatenate both parts into a single substrate document: frontmatter on top, body next.

How the skeletons map to the normative schemas:

Artifact	frontmatter (normative)	Body sections (normative)
BR	§6.5.2	§6.5.3
SR	§6.6.2	§6.6.3
TR	§6.7.2	§6.7.3
TC	§9.3	§9.4
AR	§7.4.6.2	§7.4.6.2
ACTZ	§7.13.3	§7.13.3, §7.13.5
AT	§9.19.3	§9.19.3

Conventions used in the skeletons: `<...>` is a placeholder to replace; `NN` is a sequential number within the scope; the comment `# conditional` marks a field whose obligation depends on a condition (explained inline); `# auto` marks a field maintained by the substrate/runner, not filled in by hand.

12.2 BR template

A BR captures a business need at the system or subsystem level; technical detail is prohibited in a BR (§6.5.1). Frontmatter per §6.5.2; body per §6.5.3.

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

# === Scope (mandatory) ===
level: system | subsystem              # a BR at module level is prohibited (§6.4)
scope:
  system: "<system-id>"
  subsystem: "<subsystem-id>"          # null if level=system

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

# === Source: provenance (see §7.4.1) ===
source:
  tz-section: "§N.N"                   # always mandatory — primary provenance from
the TZ
  adapt: ADAPT-NNN                     # conditional: present if an ADAPT was
created
  adapt-section: "Forward §N"          # mandatory if adapt is set
  adversarial-review-ref: AR-NNN       # conditional: if adapt is absent — the AR
carrying the "no findings" verdict (§7.4.1.2)

# === Cross-level link subsystem BR → system BR (see §6.8.2) ===
implements:                            # array; not a parent-edge, a separate edge
type
  - id: BR-NN                          # id of the parent system BR
    scope:
      system: "<system-id>"
    rationale: "<short>"               # optional; reference to an ADAPT section if
present

# === Relationship graph (substrate-managed) ===
children: []                           # auto: SR whose parent.id = this BR
implemented-by: []                     # auto: subsystem BR referencing it via
implements[]
verified-by: []                        # auto: TC verifying through SR

# === AI provenance (mandatory on RENAR-4+; schema — §4.10.1) ===
ai-provenance:
  generated-by: "<vendor>-<model>@<date>"
  generated-at: "<ISO-8601>"
  human-edits: boolean

# === Replacement (mandatory if applicable) ===
replaces: "<old-id>"
replaced-by: "<new-id>"
```

```
deprecated-date: "<ISO date>"
---
```

```
## Need

Who (role), what (action), why (business goal) — one sentence.

## Success criteria

1. <Measurable outcome, independently verifiable.>
2. <…> (3–7 items in total.)

## Context

Where the requirement came from (with a reference to an ADAPT section if
present);
what alternatives were considered.

## Constraints

<Optional: business constraints — budget, deadlines, regulation. Technical
constraints do not belong here — the SPEC types and SR exist for those.>
```

The `source.tz-section` field is always present; `source.adapt` is omitted when the adversarial review returns a "no findings" verdict — then `source.adversarial-review-ref` is mandatory (§7.4.1).

> ***Where the "requirements" live in a BR.*** *The BR template deliberately has no section with "the system shall …" statements: in RENAR those statements are SR (§6.6), derived from this BR. Mixing them into the BR blurs the "business need ↔ system requirement" boundary and produces "technical BRs" indistinguishable from SR (§6.4, §6.5.1). The BR's own verifiable, enumerable content is carried by the* ***Success criteria*** *section: 3–7 measurable, independently checkable outcomes — these are the business requirements in verifiable form. Decomposing BR → SR turns each criterion into one or more normative SR ("the system shall …" form per §6.6.3).*

12.3 SR template

An SR captures what the system does (observable behavior and constraints); table, framework, and data-structure names are the responsibility of SPEC (§6.6.1). Frontmatter per §6.6.2; body per §6.6.3.

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

# === Scope (mandatory) ===
```

```yaml
level: system | subsystem | module
scope:
  system: "<system-id>"
  subsystem: "<subsystem-id>"        # null if level=system
  module: "<module-id>"              # null if level ≠ module

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

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

# === Source: provenance (see §7.4.1; same rules as BR) ===
source:
  tz-section: "§N.N"                 # always mandatory
  adapt: ADAPT-NNN                   # conditional
  adapt-section: "Forward §N"        # mandatory if adapt is set
  adversarial-review-ref: AR-NNN     # mandatory if adapt is omitted — the AR
(§7.4.6)

# === Quality characteristic (conditional; §6.6.2) ===
# mandatory for a non-functional SR; the value comes from the ISO/IEC 25010:2023
list.
# Omitted for a functional SR.
quality-characteristic: functional-suitability | performance-efficiency |
compatibility |
                        interaction-capability | reliability | security |
maintainability |
                        flexibility | safety

# === Relationship graph ===
constrained-by:                      # typed edges to SPEC (chapter 8)
  - SPEC-UI-NN
  - SPEC-API-NN
  - SPEC-DATA-NN
children: []                         # auto: TR whose parent.id = this SR
verified-by: []                      # auto: TC verifying the SR

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

# === Replacement (mandatory if applicable) ===
replaces: "<old-id>"
replaced-by: "<new-id>"
deprecated-date: "<ISO date>"
---
```

```
## Requirement

One sentence in normative form: "The system MUST …" (modality per the convention
of §0.5).

## Behavior

A detailed description of observable behavior; functional scenarios.

## Constraints

<Mandatory if applicable: non-functional constraints — performance, security.
Full constraints are pushed into SPEC via constrained-by[].>

## Link to SPEC

<Mandatory if constrained-by[] is present: which aspects of behavior are governed
by which SPEC.>
```

`parent.id` is the single BR (a parent tree); `constrained-by[]` is a graph of references to SPEC of any type and in any number (§6.6.2).

12.4 TR template

A TR is the atomic unit of implementer work: exactly what to build within a single SR (§6.7.1). Frontmatter per §6.7.2; body per §6.7.3.

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

# === Scope (mandatory) ===
level: system | subsystem | module   # system — rare, cross-subsystem tasks
scope:
  system: "<system-id>"
  subsystem: "<subsystem-id>"        # null if level=system
  module: "<module-id>"              # null if level ≠ module

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

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

# === Source: traceability chain (inherited from parent SR, §6.7.5) ===
```

```yaml
source:
  adapt: ADAPT-NNN                  # auto: inherited from parent SR; may be
absent
  sr-version: "<version-ref>"       # pinning to the SR version (substrate
capability V5)

# === Relationship graph ===
implements-spec:                    # typed edges to SPEC
  - SPEC-API-NN
  - SPEC-UI-NN
verified-by: []                     # auto: TC verifying through SR

# === Goal and acceptance criteria ===
goal: "<one-sentence outcome>"
acceptance-criteria:
  - "<numbered, falsifiable, unambiguous>"
  - "<…>"

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

```markdown
## Goal

One paragraph; the outcome the TR makes observable.

## Acceptance Criteria

1. <Falsifiable criterion; covers a positive scenario.>
2. <Falsifiable criterion; covers a negative scenario / boundary.>

## Scope

What is in and what is **not** in the TR (per SENAR Rule 2).

## References

<Mandatory if applicable: to the SPEC in implements-spec[] and the sections of
the
parent SR.>
```

The section names `Goal`, `Acceptance Criteria`, `Scope` are canonical per §6.7.3. The TR implementer works within the SR / SPEC and does not reach the ADAPT directly (§6.7.5).

12.5 TC template

A TC verifies a normative assertion of a BR / SR / SPEC; the common frontmatter is per §9.3, the body per §9.4. Type-specific fields (`judge` , `baseline` for `ux` / `eval`) are added on top per §9.6.

```
---
# === Identity (mandatory) ===
id: TC-NN                              # immutable; NN sequential within the scope
title: "<short, descriptive>"
type: TC
slug: "<kebab-case>"

# === Classification (mandatory) ===
tc-type: business | ux | system | contract | eval | security   # business — the
canonical name (§9.5)
negative: boolean                      # true for the paired negative TC

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

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

# === Verification target (mandatory; at least one) ===
verifies:
  - id: SR-NN | BR-NN | SPEC-<TYPE>-NN
    requirement-version: "<version-ref>"   # artifact version pinning (V5)

# === Pair link (mandatory if negative=false and a paired TC exists) ===
paired-with:
  - TC-NN

# === Task binding (optional; §9.19.7) ===
verifies-tr: TR-NN                     # the task to whose scope the test is
narrowed
verifies-claims: []                    # the subset of the parent SR's claims
covered within that TR

# === Bench and data (conditional; §9.3) ===
environment-ref: SPEC-TEST-NN          # mandatory if automation.kind: dynamic —
the bench and dataset
                                       # the TC is valid against. Does not apply to
static checks
                                       # (doc-lint, structural TC): the analyzer
works over artifacts

# === Automation (mandatory) ===
automation:
  status: automated | manual-pending
  kind: dynamic | static                               # mandatory; static — the
runner is a static analyser
```

```
  location: "<substrate pointer to implementation>"  # mandatory if automated
  manual-pending-until: "<ISO date>"                  # mandatory if manual-
pending
  manual-pending-reason: "<text>"                     # mandatory if manual-
pending

# === Red history (§9.18.2; the condition for counting a TC as evidence) ===
red-history:                          # auto: maintained by the substrate from run
facts
  fixing-run:                         # the fixing run before implementation; MUST
be red
    date: "<ISO-datetime>"
    result: fail
  green-transition:                   # the recorded red → green transition
    date: "<ISO-datetime>"
  inherited-from: null                # conditional: TC-NN when behaviour did not
change (refactoring, migration)
  not-applicable-reason: null         # conditional: implementation-originated —
then a killed mutant is mandatory
  mutation-check:                     # mandatory when not-applicable-reason is
set
    mutants-killed: 0                 # >= 1, otherwise the TC is not evidence

# === Execution (mandatory for tc-type: ux | eval) ===
judge:
  vendor: "<provider>"                # mandatory; judge isolation — P7
  model: "<model-id>"
baseline:                             # mandatory for ux | eval
  artifact: "<substrate pointer>"
  perceptual-diff-threshold: float    # for ux
  metric-thresholds: {}               # for eval

# === Last run (runner-managed; not filled by the author) ===
last-run:                             # auto
  date: "<ISO-datetime>"
  result: pass | fail | skipped | n/a
  runner-id: "<runner-name@version>"

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

Context

Which clause of the verified artifact the TC references; a quote or paraphrase of the assertion.

Preconditions

The system and data state required for the run; provided by the seed mechanism.

```
## Steps

Runner actions. For tc-type: ux — intentions, not selectors (§9.6.1).

## Pass criterion

Binary, observable, reproducible (§9.11).

## Fail criterion

A list of observable signs of a violation (not the negation of the Pass
criterion):
leaks, side effects, race conditions.

## Postconditions

The state expected after the run; the cleanup mechanism.

## Out of scope

What is **deliberately** not checked, naming the paired TC where it is covered.
```

The `environment-ref` field is mandatory for dynamic TCs (`automation.kind: dynamic`): "the test passed" only means something against a specific bench and specific data (§9.3). It does not apply to static checks (the doc-lint, the structural TC) — the analyzer works over artifacts. The reference points to a SPEC-TEST (§8.5.10); changing the bench or the dataset increments its version and invalidates `verified` on the dependent TCs — precisely, without touching anything else.

The headings `## Pass criterion` and `## Fail criterion` are fixed: the change-of-criteria control hook detects them (§10.11.3) — they may not be renamed locally. The "Out of scope" section is mandatory: its absence blocks the TC transition to `ready` (§9.4).

12.6 AR template — the adversarial-review record

AR captures the fact and the outcome of the mandatory adversarial review (§7.4.6). It is an **evidence record**, not a requirements artifact: it has no parents, no children and no test cases. It is issued in both outcomes of the review — whether an ADAPT is created or not.

```
---
id: AR-NNN                          # immutable; NNN is sequential within the
parent TZ
type: AR
tz-ref: TZ-YYYY-NNN                 # mandatory; the (delta-)TZ under review
trigger-stage: import-tz            # mandatory: import-tz | decompose-br |
decompose-sr | spec | tc

# === Reviewer (mandatory; isolation §7.10.2) ===
reviewer:
  vendor: "<provider>"
```

```
  model: "<model-id>"                # MUST differ from the primary agent's model

# === Verdict (mandatory) ===
verdict: no-findings                 # no-findings | findings-present
produces-adapt: []                   # conditional: non-empty when findings-
present; empty when no-findings

# === Lifecycle (mandatory) ===
status: issued                       # draft | issued | superseded
superseded-by: null                  # conditional: AR-NNN when status=superseded

# === Signature (mandatory for issued; substrate capability V6) ===
signature:
  author: "<reviewer-id>"
  timestamp: "<ISO-8601>"
---
```

Body on `verdict: no-findings`:

```
## Justification of unambiguity

Why converting the reviewed TZ sections into requirements produces no gap: how
each risk
(terms, scope boundary, tacit assumptions) is closed.

## Review coverage

The TZ sections included in the review, and the derivation stage at which it was
conducted.
```

Body on `verdict: findings-present`:

```
## Review coverage

The TZ sections included in the review, and the derivation stage.

## Findings produced

Pointers to the `B-NNN` records in the ADAPT that was produced — **without
duplicating** their
content: the findings themselves live in the ADAPT (§7.4.4).
```

An issued AR is immutable. If a repeat review at the same stage issues a new verdict, a new AR is issued and the previous one moves to `superseded` with `superseded-by` filled in. Referencing an AR in status `draft` or `superseded` from `source.adversarial-review-ref` is prohibited — the gate reports a fatal error (§10.11.1).

12.7 ACTZ template — the TZ clarification protocol

ACTZ is an artifact of the contractual contour: a batch of questions, proposals and decisions put to the client and signed by both parties (§7.13). Decisions are worded in the language of obligations ("the button is named X"), not of interpretation. The frontmatter is per §7.13.3; TZ annexes are per §7.13.5.

```yaml
---
id: ACTZ-NNN                          # immutable; sequential within the parent TZ
title: "TZ Clarification Protocol No. N"
type: ACTZ
tz-ref: TZ-YYYY-NNN                   # mandatory; the TZ the protocol belongs to

# === Findings closed (conditional) ===
resolves:                             # B-NNN entries in an ADAPT that the
protocol closes
  - id: B-NNN                         # absent on a client-initiated protocol
(§7.13.2)
    adapt: ADAPT-NNN

# === Decisions (mandatory; non-empty list) ===
decisions:
  - number: "§M"                      # stable clause number; the target of
decided-in
    statement: "<decision in the language of obligations>"
    tz-section: "§N.N"                # the TZ section being clarified

# === TZ annexes (conditional; §7.13.5) ===
annexes:
  - name: "<annex>"
    version: "<new version>"
    document-ref: "<substrate link>"

# === Lifecycle (mandatory) ===
status: draft                         # draft | sent | signed | superseded
superseded-by: null                   # conditional: ACTZ-NNN when
status=superseded

# === Signatures (mandatory for signed; substrate capability V6) ===
client-signature:
  signed-by: "<name>"
  role: "<role of the authorised representative>"
  organization: "<client organization>"
  signed-at: "<ISO-datetime>"
vendor-signature:
  signed-by: "<name>"
  role: "<role>"
  signed-at: "<ISO-datetime>"
---
```

```
## Subject of the clarification

Which TZ sections are clarified and why the question arose — with a reference to
the TZ
section (and to the `B-NNN` finding in the ADAPT where one exists).

## Decisions

1. **§1.** <The decision in the language of obligations; a checkable wording.>
2. **§2.** <…>

Clause numbering is stable: `B-NNN.decided-in` and `AT.verifies[]` point at it.

## Annexes

<Mandatory where applicable: the new versions of the TZ annexes approved by this
protocol
(§7.13.5). The previous version remains immutable.>

## Signatures

A two-sided signature: the client (or an authorised representative) and the
vendor.
```

A signed ACTZ **MUST NOT** be edited: a correction is made only by a new protocol that supersedes the earlier decision (`superseded-by`). Referencing an ACTZ in status `draft` from `decided-in` is prohibited (§7.13.4). The effective TZ = the initial TZ with its annexes plus every signed ACTZ (§7.14).

12.8 AT template — the acceptance test of the contractual contour

An AT is derived **exclusively** from the effective TZ and checks conformance to the contract, not to the interpretation (§9.19). An AT is created by an isolated agent: only the effective TZ is given as input; access to ADAPT, BR / SR / SPEC, TC and code is prohibited, and a breach of isolation is fatal. The frontmatter and body are per §9.19.3.

```
---
id: AT-NN                           # immutable
title: "<short, descriptive>"
type: AT
negative: boolean                   # mandatory; pos/neg pairing — as for TC
(§9.7)

# === Verification target (mandatory; contractual references only) ===
verifies:
  - "TZ §N"                         # a section of the effective TZ
  - "ACTZ-NNN §M"                   # a clause of a signed protocol
  # a reference to BR / SR / SPEC / TC is fatal (§9.19.2)

tz-version: "<effective TZ revision>"  # mandatory; the revision the AT was
```

```
derived from

# === Provenance of the isolated agent (mandatory) ===
generator:
  vendor: "<provider>"
  model: "<model-id>"                # MUST differ from the primary agent's model
  internal-contour-access: false     # mandatory; confirmation of no access to
the internal contour
  generated-at: "<ISO-8601>"

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

# === Trial environment and dataset (mandatory; §9.19.3) ===
environment-ref: SPEC-TEST-NN        # set by the Architect or the runner AFTER
generation:
                                     # the isolated agent does not see internal
artifacts (§9.19.2)

# === Automation (mandatory) ===
automation:
  status: automated | manual-pending
  kind: dynamic | static
  location: "<substrate pointer to implementation>"

# === Last run (runner-managed; not filled by the author) ===
last-run:                            # auto
  date: "<ISO-datetime>"
  result: pass | fail | skipped | n/a
  runner-id: "<runner-name@version>"
  tz-version: "<effective TZ revision at the time of the run>"
---
```

```
## The effective-TZ clause

> "<The verbatim quotation of the clause under test — a TZ section or a clause of
a
> signed ACTZ.>"

This section is mandatory (`tz_text`, §9.19.3): at acceptance the clause of the
contract
itself is produced, not a paraphrase.

## Preconditions

The system and data state required for the run.

## Steps

Runner actions — worded from the client's standpoint, without relying on the
internal
```

```
construction of the system.

## Pass criterion

Binary, observable, reproducible; derived from the quoted clause.

## Fail criterion

A list of observable signs of non-conformance to the contract.

## Out of scope

What is **deliberately** not checked, naming the paired AT where it is covered.
```

ATs **are regenerated before every trial** from the current revision of the effective TZ: an outdated trial programme MUST NOT be admitted to the run (§9.19.4). The product is not submitted for hand-over until every AT is in status `passing` (§10.4.3).

12.9 SPEC templates — deferred

Skeletons for the SPEC types (chapter 8) are **deliberately not included** in this appendix. The partner review flagged the question of SPEC skeletons as open: the set of mandatory fields differs across SPEC types (UI, API, DATA, AI, SEC, INT) far more than across BR / SR / TR / TC, and fixing a skeleton prematurely risks reading as normative.

Draft sketches of the SPEC skeletons are kept in the repository — `research/17-specification-schema-and-templates.md` (§5; an internal draft, not published to the site) — pending a separate decision. When that decision is made, the SPEC skeletons are added here as a new section — without changing the closed list of SPEC types, which is already normative in `standard/08 §8.2.2` and §8.3.

The closed list now holds **eleven** types: the nine structural ones plus `SPEC-TEST` (benches and data — how conformance is proven) and `SPEC-DOC` (the composition of the delivered documentation). The deferral of skeletons covers them too: the field schemas of both types are given in `reference/02` (sections 6.3 and 6.4); document skeletons are not.

12.10 Filling placeholders and common mistakes

Placeholder	What to replace it with	Common mistake
`BR-NN` / `SR-NN` / `TR-NN` / `TC-NN` / `AT-NN`	A sequential ID within the scope (the substrate assigns it on creation)	Changing the ID after publication — it is immutable
`ACTZ-NNN` / `decisions[].number`	The protocol's sequential number and the stable clause number of a decision	Renumbering the clauses of a signed protocol — `decided-in` and `AT.verifies[]` point at them

`AT.verifies[]`	Only `TZ §N` and `ACTZ-NNN §M`	Referencing an SR / SPEC / TC — a breach of isolation; the gate reports a fatal error
`<system-id> / <subsystem-id> / <module-id>`	Identifiers from the project's system registry	Filling `subsystem` when `level=system` (it must be `null`)
`source.tz-section`	The section of the source TZ — always present	Omitting it on the assumption that a reference to the ADAPT is enough
`constrained-by[] / implements-spec[]`	IDs of existing SPEC	Naming a SPEC type outside the closed list
`# auto` fields (`children`, `verified-by`, `last-run`)	Nothing — the substrate / runner maintains them	Filling them by hand and diverging from the relationship graph

Placeholders like `BR-NN` and empty `<...>` are an unfilled skeleton, not a valid artifact: run such files through checks only after substituting real values, otherwise the substrate validators will rightly reject the template IDs.

← Back to the reference overview

Recommended TZ form

> ***Status:*** *informative. This is a* ***recommendation****, not a norm. RENAR conformance does not depend on the form of the incoming TZ in any clause: the standard accepts the TZ* ***as given*** *— including a poorly structured one — and normalizes not the writing of the TZ but the work with it (* `standard/01 §1.3` *: RENAR does not normalize the TZ authoring process on the client's side). The only normative mechanism for working with a TZ of any quality is the ADAPT contour (* `standard/07` *).*
>
> *This appendix creates no obligations and extends no closed list (§1.7.5).*

13.1 Status and boundaries

The TZ is a document external to the standard: the client writes it or brings it, it lives in the contractual contour, and it follows the vendor's business practice. The whole machinery of the standard — forward adaptation, backward findings, ACTZ — is built precisely on the assumption that an incoming TZ may be anything at all. Were the standard to require a TZ form, it would stop accepting other parties' TZs and would lose its own input condition.

Hence three hard boundaries, upheld across the corpus:

1. **Conformance does not depend on the TZ form.** The conformance chapter (`standard/13`) does not mention the form of the incoming TZ in any clause. An organization whose TZ has an arbitrary structure may claim full conformance.
2. **Substrate checks do not touch the TZ structure.** No automated check parses a TZ into sections or requires that any be present. Requiring a TZ structure through an automated check would violate the boundary of §1.3.
3. **The TZ does not become a requirements registry.** Duplicating BR / SR into the TZ is an explicit anti-recommendation (§13.5).

The only admissible argument in favour of the form is an observation, not an obligation: a TZ written to this form produces fewer backward findings, shortens the clarification rounds, and makes the derivation of acceptance tests reliable (§13.6). The choice remains with the vendor and the client.

13.2 The basis of the form: an inversion of the finding typology

The form is recommended not out of one vendor's habit but on a basis internal to the standard. The seven categories of backward findings (§7.4.4) are, in essence, a catalogue of typical TZ defects. Therefore the **recommended form is an inversion of the finding typology**: each section exists not for its own sake but as prophylaxis against a specific category of defect.

Section of the form	Category of findings it forecloses
Out of scope	`scope` — an unclear boundary of work
User roles (explicit authorization)	`hidden-assumption` — an assumption that may be wrong
Integrations (constraints, risks)	`feasibility` , `regulatory`

Definitions	`terminology` — direct prophylaxis
Key data entities	`terminology` (entities), `hidden-assumption`
"Given — when — then" criteria on every functional requirement	`gap` — the TZ is silent about something without which implementation is impossible
Use-case scenarios with pre- and postconditions	`contradiction` — scenarios collide requirements against one another
Accompanying artifacts (a table)	addressability of annexes: each annex becomes a unit that can be referenced

The table also works in reverse — as an instrument for finding holes in the form itself. It is precisely this table that exposed the fact that the `terminology` category, until the "Definitions" section appeared, was the only one without a prophylactic section.

13.3 Sections of the form

Eleven sections. The numbering is internal to the TZ; the stable identifiers of clauses (`UC-NNN` , `FR-NNN` , `NFR-NNN`) are assigned by the client and do not change afterwards: acceptance tests reference them (§9.19.3).

#	Section	Content
1	General description	One paragraph: what this is, for whom, how it works
2	**Definitions**	The client's terms with fixed meanings (§13.4)
3	User roles	A table of roles; authorization is stated explicitly — including an explicit "not provided"
4	Use-case scenarios	`UC-NNN` : participant, precondition, steps, postcondition
5	Functional requirements	`FR-NNN` : priority and "given — when — then" acceptance criteria, including at least one negative variant with invalid data
6	Non-functional requirements	`NFR-NNN` : only what is explicitly constrained or significant
7	Integrations	A table per external system: address, type, constraints, data in and out, risks
8	Key data entities	A vocabulary of the problem domain — not a database schema
9	Accompanying artifacts	A table of annexes: type, coverage, location, status
10	Out of scope	An explicit list of what is not included
11	Project acceptance criteria	Conditions of acceptance, the reference scenario

Sections 4 and 5 are what the acceptance-test programme is derived from (`reference/14`). Section 11 is its embryo: from the outset the client sees how the handover will proceed.

13.4 The "Definitions" section — second, and why exactly there

The section is placed **second**, immediately after the general description and before the first use of terms in scenarios and requirements. This follows GOST and ISO practice, where terms and definitions come at the start of a document.

The content is the client's terms that admit a second reading: roles and their synonyms, action verbs ("post", "approve", "export"), statuses, units of measure, the organization's abbreviations. The form is a table: "term — meaning — synonyms, or what the term is not".

Term	Meaning	Synonyms / what it is not
Post a document	Commit a document to the ledger such that it affects balances	Not "save": a draft does not change balances

Two recommendations for filling it in:

- include only terms with a possible second reading — the section is not a substitute for a dictionary of ordinary words;
- define a term with a self-contained formulation. Definition by usage ("as in section 4") is inadmissible: it does not remove the ambiguity, it hides it.

The section is a direct input for term mapping in ADAPT: a term fixed by the client produces no finding of the `terminology` category, and the whole terminological layer need not be reconstructed through rounds of clarification, when the client could have fixed their own terms up front.

13.5 What a TZ is not

The TZ remains a document in the language of the client and their problem domain. The form structures the client's language — it does not turn the TZ into a requirements registry.

Duplicating BR / SR into the TZ is **not recommended**. The requirements of the standard live in the internal contour and are derived from the TZ through interpretation (§7.12); copied into the TZ, they diverge from the original at the first edit and produce two sources of truth instead of one. Decoupling the contours is not a formality but a condition for traceability to work.

The distance between the language of the TZ and the language of the standard is variable and normal (§7.12.1). The form shortens that distance; it does not abolish it.

13.6 The observed effect

The effect is recorded as an observation — neither a promise nor an obligation:

- fewer backward findings — each section of the form forecloses its own category in advance (§13.2);
- a shorter path to requirements — fewer ACTZ clarification rounds (§7.13);
- a more reliable derivation of acceptance tests: stable `UC-NNN` and `FR-NNN` identifiers give the `verifies` field its addressing without interpretation, while the mandatory negative variant in the acceptance criteria yields positive/negative pairing of acceptance tests almost for free (§9.19).

A vendor with their own TZ loses only speed, not conformance.

13.7 Relation to other documents

Document	Relation
`standard/07 §7.4.4`	The seven categories of backward findings — the basis of the form
`standard/07 §7.13`	ACTZ — the TZ clarification protocol
`standard/07 §7.14`	The effective TZ — the benchmark for acceptance
`standard/09 §9.19`	AT — the acceptance test of the contractual contour
`reference/14`	Recommended form of the acceptance-test programme
`reference/12`	Templates for internal artifacts (BR / SR / TR / TC / AR / ACTZ / AT)

← Back to the reference

Recommended form of the acceptance-test programme

> ***Status:*** *informative. The recommendation concerns the* ***human-readable document produced for the client****. The structure of the AT artifact itself is normative and is set by* `standard/09 §9.19` *; where a skeleton and a chapter disagree,* ***the chapter prevails****.*
>
> *This appendix creates no obligations and extends no closed list (§1.7.5).*

14.1 What this document is

The acceptance-test programme is the scenario document produced for the client at handover. It is the **client-facing projection of the set of ATs**: the machine executes the acceptance tests, the human reads the programme. That split continues a general principle of the standard: the client sees the human-readable, the machine works with the canonical.

The programme is **derived from the effective TZ** (§7.14) — the initial TZ with its annexes plus every signed ACTZ — and not from the initial TZ. The contract lives incrementally, and a system cannot be produced against a stale revision: ATs are regenerated before every trial from the revision in force (§9.19.4), and the programme is regenerated after them.

The programme is generated, not hand-written: everything in it already exists in the ATs.

14.2 Fields of the programme

Field	Content	Source in the standard
`id`	The stable identifier of the contract clause (`UC-NNN` , `FR-NNN`) or of an ACTZ clause. Not invented — taken from the document	`reference/13 §13.3`
`source`	Where the clause comes from: `TZ §N` or `ACTZ-NNN §M` , including subsystem TZs	`verifies[]` , §9.19.3
`tz_text`	The **verbatim quotation** of the effective-TZ or ACTZ clause	a mandatory body section of an AT, §9.19.3
`steps`	Verification steps as active commands: "open…", "click…", "confirm…"	AT body
`expected`	The expected result in one sentence	AT body

An example of a single programme entry:

```
id: FR-014
source: "TZ §5.14"
tz_text: >
```

```
  The system shall reject posting of a document when the sum of the
  line items diverges from the sum stated in the header.
steps:
  - "Open a document whose header sum diverges from its line items."
  - "Click «Post»."
  - "Confirm the document is not posted and the reason for refusal is shown."
expected: "The document remains a draft, and the reason for refusal is stated
explicitly."
```

The verbatim quotation is the key field. The client sees the text of the contract and the verification side by side: what is produced is not a paraphrase of the clause but the clause itself. There is nothing left to argue about regarding "what was meant" — which is why the quotation was taken into the normative part of the AT rather than left as a recommendation.

14.3 Coverage completeness

Every contract clause enters the programme — with no omissions and with no merging of several clauses into one. Completeness is counted over the sections of the effective TZ and the clauses of the signed ACTZ; the machine-side counterpart of this requirement is the `ACCEPTANCE.md` report (§9.19.6).

Merging clauses looks like a saving, but it removes addressability: the failure of a merged check cannot be attributed to a specific contract clause, and the verdict once again becomes a matter of discussion.

The programme may be agreed with the client through a further ACTZ (§7.13) — the programme then becomes an approved commitment in its own right, and the order of handover is known to both parties in advance.

14.4 The verdict by defect class

The acceptance verdict is best computed **by defect class rather than as a binary**. The mechanism: every failed programme entry is classified, and the decision to sign the acceptance certificate is taken according to quality corridors declared in advance.

A typical gradation has four levels:

Level	Name	Description
1	Critical	The system is inoperable, data is lost or corrupted, security is breached
2	Major	Key functionality is unavailable or works incorrectly, with no workaround
3	Moderate	Secondary functionality is incorrect, a workaround exists
4	Minor	Visual, textual, and other defects that do not affect operation

> ***The gradation is given as an example, not as a norm.*** *The standard establishes **neither** numeric thresholds **nor** the number of levels itself. The quality corridors — which levels block signature of the certificate, how many defects of each level may be carried over — are declared by the **vendor** in the contract or in the manifest, based on their internal quality regulations. A bank and an in-house department will have different thresholds, and both will be right: this is commercial policy, not a property of requirements engineering.*

What is recommended regardless of the thresholds chosen: a defect that blocks signature is recorded with a reference to the contract clause (`id` , `tz_text`), a description of the divergence, and the expected result; non-blocking defects are recorded, and the plan for fixing them is agreed with the client before the certificate is signed.

The value of the mechanism lies in the rules of the game being declared before the trials. Without it, every defect found at acceptance becomes a matter of bargaining.

14.5 The internal gate and contractual acceptance are not the same thing

Quality corridors describe the behaviour of the parties **at acceptance** and do **not** weaken the vendor's internal gate.

Contour	Rule	Who finds the defects
Internal, before handover	Every AT in `passing` status — strictly, with no corridors (§10.4.3)	The vendor
Contractual acceptance	The verdict follows the vendor's quality corridors	The client

A vendor does not produce a system with known failures — otherwise the corridors turn into permission to hand over unfinished work. But at a real acceptance the client finds their own defects, and the classification supplies rules agreed in advance for that situation.

14.6 The defect level as diagnostics

The defect level correlates with the routing of AT and TC failures (§9.19.5):

- a level 1–2 defect with green TCs is almost always an **interpretation error**: the system conforms to ADAPT but not to the contract. It is fixed in ADAPT and ACTZ, not in the code;
- level 3–4 defects more often point to under-specified detail — candidates for a further ACTZ.

14.7 Relation to other documents

Document	Relation
`standard/09 §9.19`	AT — the normative structure of the artifact the programme is derived from
`standard/07 §7.13`	ACTZ — the protocol for clarifying the TZ and agreeing the programme
`standard/07 §7.14`	The effective TZ — the benchmark the trials run against
`standard/10 §10.4`	The internal release gate
`reference/13`	Recommended TZ form — the source of stable clause identifiers

← Back to the reference