

RENAR

Practical guides for adopting RENAR

RENAR · Practical Guide · Version 1.1 | 19.09.2026

Authors: Vadim Soglaev, Andrey Yumashev

CC BY-SA 4.0 | renar.tech

00. Quickstart

End-to-end example: a small "email/password sign-up" project. The full RENAR cycle with a minimal set of artifacts: TZ → ADAPT → BR → SR → SPEC → TC. Time: ~30 minutes of reading + sketching on paper; ~2-3 hours if you do it live on a substrate. Prerequisites: core/renar-core.md (≤ 10 min). The full-size example with 2FA — 01-walkthrough.md. The RU corpus language — standard/00 §0.7.

The steps below are one possible order of work, chosen for a small example; only the preconditions and rules of the standard are normative ([standard/01 §1.2.1](#)) — your organization MAY arrange the order its own way.

After this start you will have: an understanding of the full RENAR cycle on one small example; ready-made YAML templates for each artifact type; experience passing two gates (QG-ADAPT-approve ≡ **QG-3 Architecture Gate** §10.4.1 + **QG-2 Verification Gate**); a point from which to scale.

Prerequisites

Substrate (`substrate`) — the system for storing and versioning artifacts. RENAR is independent of the kind of store; for the quickstart, any substrate with V1-V6 capabilities will do ([reference/01 §2.7](#)): git with PR review; a document-oriented store; a DBMS with history and signatures.

Folder structure (the default convention for git):

```
my-project.req/  
  tz/                # immutable client TZ  
  actz/              # ACTZ – TZ clarification protocols (dual signature)  
  adapt/             # ADAPT artifacts  
  br/                # Business Requirements  
  sr/                # System Requirements  
  specs/{arch,api,data,ui,sec,ai,proc,int,ops,test,doc}/  
  tests/             # TC (tc-type incl. contract)  
  at/                # AT – acceptance tests derived from the effective TZ
```

For other substrates, use an equivalent organization by document type or namespace.

Step 1. TZ (5 min)

The client provides a small TZ. It is a **contractual immutable** document — once signed, it is not edited.

tz/TZ-2026-001.md (excerpt):

```
---  
id: TZ-2026-001  
title: "User registration via email"  
signed-date: "2026-05-15"
```

```
signed-by-client: "J. Ivanov, PM, ClientCo"
```

```
---
```

```
# TZ-2026-001
```

§1. Context

Build a system for user registration via email.

§2. Requirements

- A user can register via email and password.
- After registration, the user confirms their email.
- After confirmation – access to the personal dashboard.

§3. Constraints

- Web application only.
- Storage in the RF (03-152).

The TZ is signed by the client → immutable. All edits and interpretations go through ADAPT.

Step 2. ADAPT draft (10 min)

Create `adapt/ADAPT-001-main.md`. An AI agent or an engineer fills in **Forward** (how we understood it) and **Backward** (what is unclear).

```
---
id: ADAPT-001
title: "Adaptation of TZ-2026-001 – Registration via email"
type: ADAPT
source-tz: { id: TZ-2026-001, signed-date: "2026-05-15", signed-by-client: "J.
Ivanov, PM, ClientCo", document-version-ref: "<version id>" }
status: draft
created: "2026-05-16"
ai-provenance: { generated-by: "anthropic-claude-opus-4-7@2026-05-16", prompt-
template: "prompts/adapt-from-tz.md@v1.0", context-tokens: 1024, output-tokens:
2048, human-edits: false }
---
```

Forward §2 Requirements:

****Quote:**** "A user can register via email and password. After registration, the user confirms their email. After confirmation – access to the personal dashboard."

****Interpretation:**** POST /auth/sign-up with {email, password}. A User is created with status `unverified`. A verification link email is sent. Click → status `verified`. Only `verified` users can sign in.

****Elaborated scenarios:**** sign-up; email verification; first sign-in.

****Coverage:**** in scope – email/password + verification + dashboard access. Out of scope: OAuth, SMS, 2FA, password reset.
****Forward links**** (auto-populated after approval): BR-01, SR-01, SR-02, SR-03.

Backward (3 entries):

B-001: gap – sign-in attempt by an unverified user

Status: open. Description: the TZ does not describe system behavior on a sign-in attempt before email verification.

Question to client: show a "verify your email" message with a resend button – or a 401 with no explanation?

B-002: regulatory – 03-152 storage in the RF

Status: open. Description: TZ §3 requires "storage in the RF". What exactly: the data center, the provider's jurisdiction, a separate installation?

Question to client: clarify the scope of 03-152.

B-003: gap – resending the email

Status: open. Description: what if the verification email is lost? what about a rate limit?

Question to client: the resend policy.

Step 3. ACTZ and ADAPT approved (5 min)

Questions to the client do not go "into the email thread": the Architect collects them into an **ACTZ** — an agreed clarification of the TZ (in contractual usage, "TZ Clarification Protocol No. 1"). An ACTZ is signed by both parties, and it is the ACTZ that carries contractual weight ([standard/07 §7.13](#)). The backward lifecycle:

open → asked-to-client → answered → resolved → frozen (after approval) ; the protocol lifecycle: draft → sent → signed .

actz/ACTZ-001.md — the client's decisions, phrased in the language of obligations:

```
---
id: ACTZ-001
title: "TZ Clarification Protocol No. 1 for TZ-2026-001"
type: ACTZ
tz-ref: TZ-2026-001
resolves: [B-001, B-002, B-003]
status: signed
client-signature: { signed-by: "J. Ivanov", role: PM, organization: "ClientCo",
signed-at: "2026-05-18T15:30:00Z" }
vendor-signature: { signed-by: "P. Petrov", role: architect, organization:
"VendorCorp", signed-at: "2026-05-18T15:40:00Z" }
---
```

§1. Sign-in before email confirmation

The system shows a "confirm your email" message and a resend button. A refusal

with no explanation is not allowed.

§2. Data storage

Data is stored in a data center on RF territory; the jurisdiction is the RF. The choice of provider is left to the contractor.

§3. Resending the email

No more than once every 5 minutes and no more than 5 times per day.

The signed protocol comes back into the ADAPT: each finding receives a **decided-in** — a pointer to a clause of a **signed** ACTZ (§7.4.5). Without such a pointer a finding cannot become **resolved**.

B-001: resolved (decided-in: ACTZ-001 §1)

→ a scenario added to Forward §2; SR-04 created.

B-002: resolved (decided-in: ACTZ-001 §2)

→ data-residency: ["RU"] in Forward §2.

B-003: resolved (decided-in: ACTZ-001 §3)

→ rate-limit rules in Forward §2; SR-04 refined.

All backward entries → **resolved**, **open-questions-count** = 0. **QG-ADAPT-approve** (≡ QG-3) — all preconditions of §10.8.3 are met. The ADAPT is an internal interpretation document, so it is signed by the **Architect alone**; the client's signature lives on the ACTZ (§7.5):

status: approved

approval:

```
# no client-signature: an ADAPT is an interpretation, not an obligation
architect-signature: { signed-by: "P. Petrov", role: architect, signed-at:
"2026-05-18T16:00:00Z" }
ai-provenance: { human-edits: true }    # the architect edited the AI draft
open-questions-count: 0
resolved-questions-count: 3
```

After approval the ADAPT is **immutable** (frozen). All BR/SR/SPEC reference the approved ADAPT, not the TZ directly.

The **effective TZ** = the initial TZ + all signed ACTZs (§7.14). It is against the effective TZ — not against the ADAPT — that the system is later delivered and accepted.

Step 4. BR-01 (3 min)

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

```
---
id: BR-01
```

```

title: "User registration via email"
type: BR
priority: must
source: { adapt: "ADAPT-001", adapt-section: "Forward §2", tz-section: "§2" }
business-context:
  stakeholder: "J. Ivanov (PM, ClientCo)"
  business-goal: "Enable users to create an account and gain access to the
product"
business-outcome:
  measurement-type: kpi
  kpi-name: "registration-conversion-rate"
  measurement-method: "registered / visited_signup * 100%"
  baseline-value: 0
  target-value: 60
  target-met-by: "2026-09-01"
data-classification: { contains-pii: true, retention-days: 2555, data-residency:
["RU"] } # 7 years per 03-152
compliance: [{ standard: "03-152", article: "ст.6,12" }]
ai-provenance: { generated-by: "anthropic-claude-opus-4-7@2026-05-18", human-
edits: true }
---

# BR-01: User registration via email

A user MUST be able to create an account on their own via email/password
and gain access to the personal dashboard after email confirmation.

```

Step 5. SR-01..SR-04 (3 min)

Decompose the BR into verifiable SRs. Each SR references the approved ADAPT.

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

```

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

## Description
POST /auth/sign-up accepts {email, password}.
- Valid data → User status `unverified`, a verification email is sent, 201.
- Invalid email (regex/format) → 422 with the field indicated.
- Email already taken → 409.
- Weak password (< 8 chars or in the blacklist) → 422.

```

Likewise — SR-02 (email verification), SR-03 (sign-in for verified users), SR-04 (email resend with the rate limit from B-003).

Step 6. SPEC-* (3 min)

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

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

Likewise — SPEC-DATA-01 (the User schema), SPEC-SEC-01 (the auth model + Φ3-152 controls).

Step 7. TC pos/neg pairing (5 min)

Each SR — at least 1 positive + 1 negative TC. The canonical schema — [reference/02 §8](#).

tests/TC-01-signup-success.md (positive, for SR-01):

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

## Given
- a new email (uniquely-generated@test.com); a valid password (length ≥ 8, not in
the blacklist).

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

```
## Then
- status 201; body {"user_id": "<uuid>", "status": "unverified"}; a User in the
DB with status `unverified`; a verification email sent (mock).
```

tests/TC-02-signup-invalid-email.md (negative, for SR-01):

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

## Given
- email "not-an-email" (no `@`); any valid password.

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

## Then
- status 422; body {"field": "email", "error": "invalid format"}; no User created
in the DB; no email sent.
```

Likewise, pairs for SR-02, SR-03, SR-04. For SR-04 (with the rate limit) — an additional TC that checks the block after the 5th attempt within 24 hours.

Step 8. Approve the set version, run the TCs, record the verification (2 min)

8.1 QG-0 of the set. BR-01, SR-01..04, three SPECs and all TC norms sit in the set draft. They have no status of their own ([standard/10 §10.7](#)) — the set is approved as a whole: every statement has a pair of TC norms, the adversarial review passed, the Architect signs. A version record is released ([§10.5.4](#)):

```
# renar/VERSIONS/1.0.md
set-version: "1.0"
major: false
approved-by: "I. Ivanov"
approved-at: "2026-05-19T09:00:00Z"
resolves-to: "<substrate-native pointer to the snapshot>"
changes: { added: [BR-01, SR-01, SR-02, SR-03, SR-04, SPEC-API-01, SPEC-DATA-01,
SPEC-SEC-01, TC-01, TC-02, "..."] }
```

8.2 QG-1 and the run.

The TC implementations are written against version `1.0` (`automation.set-version: "1.0"`) and admitted to runs; the test runner on the substrate runs them and updates `last-run`:

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

8.3 QG-2 of the version. When **all** TCs of version `1.0` are green with `last-run.set-version = 1.0` and each has a recorded red → green transition ([standard/09 §9.18.2](#)) → the engineer's spot-check ([§9.14](#)): a sample of 5 passed TCs, directed by machine signals (TCs without red history first), checked against the SR. If the spot-check passes → **QG-2 Verification Gate** passed → the runner appends the verification entry to the version record:

```
# renar/VERSIONS/1.0.md (appended by the runner only):
verification:
- { product-version: "<pointer to the build>", date: "2026-05-19T14:00:00Z",
  result: pass, runner-id: "test-runner@1.0", evidence-refs: ["<run>"] }
```

SR-01..04 are "verified" not by themselves but as part of the pair (version 1.0, build) — they have no separate status. Version 1.0 is ready to be presented.

Conformance to the **contract** is checked by a separate layer — **AT** ([standard/09 §9.19](#)): acceptance tests that an isolated agent derives from the effective TZ alone, without looking into the ADAPT, the SRs, or the code. TCs prove that the system matches the interpretation; ATs prove that the interpretation matches the contract. Until every AT is green, the product is not submitted for delivery ([§10.4.3](#)).

Traceability chain

At any moment the provenance of an artifact can be reconstructed:

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

AT-01
└─ verifies TZ-2026-001 §2 + ACTZ-001 §1 # the contract only, no internal
artifacts
```

An audit a year later: "where did the requirement to return 422 on an invalid email come from" — TC-02 → SR-01 → ADAPT-001 §2. The trace is complete.

What you just did

- Created an immutable contractual artifact (the TZ).
- Went through two-way interpretation via ADAPT with three backward findings → put the questions to the client as an ACTZ protocol → the signed decisions came back into the ADAPT via `decided-in` → approved.
- Obtained the effective TZ (the initial TZ + the signed ACTZ-001) — the benchmark for the future delivery and acceptance.
- Decomposed it into a BR + 4 SRs, bound to the approved ADAPT.
- Described 3 SPECs (API, DATA, SEC) as the parallel structural axis.
- Covered each SR with a pos+neg TC pair (at least 8 TCs).
- Passed three quality gates: QG-ADAPT-approve ($\equiv$ QG-3 Architecture), QG-0 of the set (version 1.0) and QG-2 Verification Gate (verification entry of version 1.0).

This is the full RENAR cycle in minimal form. On a real project — more BR/SR/SPEC, more backward findings, delegation to AI agents; optionally QG-4 (acceptance).

In real work, steps 2-7 are performed by an AI agent. The engineer does not write the frontmatter and body of artifacts line by line — they frame the task, read the result, refine it, and approve. This is the regular mode ([standard/00 §0.2.1](#)). The full scenario of an initial TZ and a delta-TZ with an adversarial reviewer — in [01-walkthrough.md phase 2 + phase 8](#).

What next

You want to...	Document
A detailed end-to-end example (login + 2FA, phases 0–9)	01-walkthrough.md
Transition from a legacy approach to RENAR	02-transition-guide.md
Git as a substrate (commit policy, PR review, hooks)	03-tool-guide-git.md
A document-oriented substrate	04-document-store-substrate.md
A comparison of RENAR with SAFe / BABOK / ISO 29148	05-safe-comparison.md
Compliance: GDPR / Φ 3-152 / AI Act	06-compliance.md
Failure modes — typical failures and patterns	07-failure-modes.md
The full normative specification (16 chapters)	standard/
Artifact schemas and validation rules	reference/02-schemas.md
Glossary and mapping to industry standards	reference/01-glossary.md

01. Walkthrough: Login Flow for AcmeCorp

One full RENAR cycle from a signed TZ to an accepted release. The example is an internal tool with registration via corporate email and 2FA. The goal is to show **every phase** on a single medium-sized project.

Context: AcmeCorp, ~1 sprint of team work, stack Next.js + FastAPI + PostgreSQL. RENAR maturity at the RENAR-3+ level (full ADAPT + TC + adversarial). The example is **substrate-independent**: operations go through the V1–V6 capabilities; the concrete directory layout is in 03-tool-guide-git or 04-document-store-substrate.

Prerequisites: 00-quickstart, core/renar-core, reference/01-glossary.

The phases are an illustration, not a norm. The order of phases 0–9 is one possible order of work; the standard governs preconditions and rules, not the sequence ([standard/01 §1.2.1](#)). An organization with a different process conforms to the standard as long as the preconditions of every artifact hold.

Reader's route. Phases 0–2 — context gathering, signing the TZ, the ADAPT and the TZ clarification protocol (ACTZ). Phases 3–4 — decomposition into BR/SR/SPEC and generation of pos/neg pairs for TC. Phase 5 — the canonical gates QG-0 (requirement approval) and QG-1 (the TC **draft** → **ready** transition only). Phases 6–7 — TR implementation and verification (QG-2). Phases 8–9 — a delta-TZ on changes and acceptance: AT against the effective TZ plus the optional QG-4 on the business outcome.

Phase 0 — Requirements elicitation

Before signing the TZ. The AI agent runs 2–3 interviews with stakeholders (Sales Director, IT Manager) and gathers context in structured form.

Phase 0 artifacts (informative, not normative for RENAR Core): `elicitation/{domain-context.md, sales-director.yaml, it-manager.yaml, findings-clustered.md, critic-review.md, multi-model-diff.md}`. Phase 0 is not fixed in Core — it belongs to the elicitation methodology, out of scope for RENAR v1.0 ([standard/01 §1.3](#)).

Phase 1 — TZ import

After the elicitation iterations, the client signs `TZ-2026-042` :

```
# TZ-2026-042 – Login Flow for AcmeCorp Internal Tool
```

```
Signing date: 2026-05-03 • Parties: AcmeCorp + VendorCorp
```

```
## §1. Goals
```

```
Cut AcmeCorp employees' time to enter the tool to <2 minutes from first arrival to full access.
```

```
## §2. Functional requirements
```

```
### FR-001. Registration by corporate email
```

```
An employee registers via an email in the @acmecorp.com domain. An email outside
```

the domain – denial with an explanation.

FR-002. Two-factor authentication (TOTP)

After registration, mandatory 2FA setup via TOTP.

FR-003. Access recovery via the corporate administrator

On loss of the 2FA device – recovery via a ticket in IT support.

§3. Non-functional requirements

NFR-001. Performance: Login <2 seconds (p95).

NFR-002. Security: bcrypt cost-factor ≥ 12; login logs – 1 year; lockout after 5 failed attempts in 15 minutes.

NFR-003. Jurisdiction: all data in the RU (government contracts).

The TZ is signed → **immutable**. Any edits go through ADAPT (Phase 2) or a delta-TZ (Phase 8). The runtime MUST register the immutable TZ as a revision (V1+V2) with AI provenance (V6).

Phase 2 — ADAPT and ACTZ (interpretation and the clarification protocol)

This is where the two loops part ways. The **ADAPT** is the internal interpretation: its audience is the engineer, the agent, and the reviewer, and it is signed by the Architect ([standard/07 §7.5](#)). The **ACTZ** is the TZ clarification protocol: whatever was put to the client and signed by both parties — that is, an obligation ([§7.13](#)). The boundary runs along the audience: what is shown to the client and approved is an obligation; what is not shown is an interpretation.

2.1 The primary agent generates a draft ADAPT. Input: TZ-2026-042 (immutable). Output: draft ADAPT-001 + Forward sections (across §2 FR + §3 NFR) + Backward findings (6 candidates) + V6 provenance.

2.2 Adversarial review. A separate critic agent (a different model) checks the backward findings; it blocks adapt-approve while critical findings remain open. Examples:

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

2.3 Iterative resolution. The Architect adjusts the Forward and Backward, the AI regenerates. After 2 adversarial cycles: 7 backward entries (B-001..B-007), all reclassified or prepared to be put to the client; the Forward covers §2 + §3 of the TZ in full.

2.4 ACTZ-001 — TZ Clarification Protocol No. 1. Five findings out of seven require a decision from the client. The Architect does not forward the agent's raw output: they aggregate the questions, rephrase them in the language of obligations ("the domain is compared case-insensitively"), attach proposals — and put them forward as a single protocol. The client signs; so does the contractor.

```
---
id: ACTZ-001
title: "TZ Clarification Protocol No. 1 for TZ-2026-042"
```

```

type: ACTZ
tz-ref: TZ-2026-042
resolves: [B-001, B-002, B-004, B-006, B-007]
status: signed
client-signature: { signed-by: "A. A. Ivanova", role: "Product Lead",
organization: "AcmeCorp", signed-at: "2026-05-04T11:30:00Z" }
vendor-signature: { signed-by: "P. P. Petrov", role: architect, organization:
"VendorCorp", signed-at: "2026-05-04T11:50:00Z" }
---

## §1. Email case
`@AcmeCorp.com` and `@acmecorp.com` denote the same employee: the domain is
compared case-insensitively.

## §2. The term "employee"
An employee is an account with the role `employee` in the corporate directory.

## §3. The lockout boundary
5 failed attempts within 15 minutes are counted per "IP + email" pair, not per
session.

```

Every finding that requires a decision from the client receives a `decided-in: ACTZ-001 §M` — a pointer to a clause of the **signed** protocol. A finding cannot become `resolved` without such a pointer, and an ADAPT cannot be approved while findings remain unclosed (§7.4.5).

2.5 ADAPT in status `approved` . The signature is the Architect's alone: the client never saw this document and never signed it — their decisions live in ACTZ-001.

```

---
id: ADAPT-001
title: "Adaptation of TZ-2026-042 – Login Flow AcmeCorp"
type: ADAPT
trigger-stage: import-tz
source-tz: { id: TZ-2026-042, signed-date: "2026-05-03", signed-by-client:
"AcmeCorp PM" }
status: approved
approval:
  # no client-signature per §7.5 – the ADAPT is not put to the client
  architect-signature: { signed-by: "P. P. Petrov", role: architect, signed-at:
"2026-05-04T12:00:00Z" }
generates-requirements: [BR-01, BR-02, SR-01, SR-02, SR-03, SR-04, SR-05, SR-06,
SR-07]
generates-specs: [SPEC-UI-01, SPEC-API-01, SPEC-DATA-01, SPEC-SEC-01]
open-questions-count: 0
resolved-questions-count: 7
ai-provenance: { generated-by: "anthropic-claude-opus-4-7@2026-05-04", prompt-
template: "prompts/adapt-from-tz.md@v2.1", human-edits: true }
---

```

Once approved, ADAPT-001 is **immutable**.

The **effective TZ** (§7.14) at this point: TZ-2026-042 + ACTZ-001 . That is the benchmark the system will be delivered against in phase 9. The ADAPT is not part of the acceptance benchmark — the client answers only for what they signed.

Phase 3 — Decomposition into BR / SR / SPEC

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

3.2 Adversarial findings:

```
[HIGH] BR-01 stakeholder field empty – who owns the business goal?
[HIGH] SR-05 says "bcrypt cost-factor 12" – this is a deployment detail, it
belongs in SPEC-SEC-01, not in the SR.
[MEDIUM] NFR-003 (jurisdiction) is not reflected in the data-classification of
SR-01.
[MEDIUM] SPEC-UI-01 has no accessibility-level – WCAG-AA minimum for a corporate
tool.
→ 4 findings → fix → re-generate.
```

3.3 Final artifact set:

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

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

```
---
id: SR-01
title: "Email domain validation at registration"
type: SR
```

```

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

## Description
Registration is allowed only if the email belongs to the `@acmecorp.com` domain. Other domains are rejected with an explanation.

## Behavior
- email NOT from `@acmecorp.com` → 422 with `{"error": "email-domain-not-allowed", "allowed-domain": "acmecorp.com"}`. [ADAPT-001 Forward §2.1; TZ-2026-042 §2 FR-001]
- email from `@acmecorp.com` → standard registration (SPEC-API-01).
- The whitelist is stored in `SPEC-SEC-01.allowed-domains`, extended without a release.
- Domain comparison is case-insensitive (ADAPT-001 Forward §2.1).

## Constraints
- `*.acmecorp.com` subdomain – a separate Architect decision (out of scope).

```

3.5 SPEC-API-01 (fragment):

```

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

## Endpoints

### POST /auth/register
- body: `{"email": "<corp-email>", "password": "<strong>"}`
- 201 → `{"user_id": "<uuid>", "verified": false, "totp_setup": false}` · 422 → invalid · 409 → email exists

### POST /auth/login

```

```

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

### POST /auth/totp-setup – see SR-02.

## Error model
A single structure: `{"error": "<code>", "details": {...}}`.

```

Phase 4 — TC generation (pos/neg pairs)

Operation: `tc-generate` SR-01 → pos/neg TC pairs per testable assertion (standard/09 §9.7: for each assertion in an SR — at least one positive + negative TC pair).

TC-001 (positive):

```

---
id: TC-001
title: "Registration with an allowed domain – happy path"
type: TC
tc-type: system
verifies: [{ id: SR-01 }]
negative: false
automation: { status: automated, location:
"tests/auth/test_registration.py::test_allowed_domain_succeeds", runner: pytest }
---

## Context
SR-01, "Requirement" section: registration with an email from a whitelisted
domain creates an account.

## Preconditions
- The DB is empty; email alice@acmecorp.com is not registered.

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

## Pass criterion
- status 201; body contains {"user_id": "<uuid>", "verified": false,
"totp_setup": false}; the User is created in the DB; a verification email is sent
(mock SES).

## Fail criterion
- status ≠ 201; plaintext password in the body/logs; a User with a different
email (case mismatch); the verification email is not sent.

## Postconditions
- The User is removed by the seed mechanism; the mock SES queue is cleared.

```

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

TC-004 (negative):

```
---
id: TC-004
title: "Registration with a disallowed domain – denial with an explanation"
type: TC
tc-type: system
verifies: [{ id: SR-01 }]
negative: true
automation: { status: automated, location:
"tests/auth/test_registration.py::test_disallowed_domain_rejected", runner:
pytest }
---

## Context
SR-01, "Behaviour" section: an email outside the whitelisted domain is rejected
with an explainable reason.

## Preconditions
- email "bob@gmail.com" (outside the whitelist).

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

## Pass criterion
- status 422; body == {"error": "email-domain-not-allowed", "allowed-domain":
"acmecorp.com"}; the User is NOT created in the DB; no email is sent; an audit
entry about the rejected attempt (for SR-06).

## Fail criterion
- status ≠ 422; the User is created; an email is sent (security leak); the audit
entry is missing.

## Postconditions
- The audit entry is removed by the seed mechanism.

## Out of scope
- The audit entry format → TC-021 (SR-06).
```

Phase 5 — Quality gates before code (QG-0 and QG-1)

After generating TCs for all 7 SRs — 26 test-case entries in total (pos/neg pairs + extra negatives for SR-05, SR-06).

5.1 QG-0 — approval of set version 1.0. BR-01, seven SRs, the SPECs and 26 TC norms are approved as one version (standard/10 §10.5.2). Preconditions: `source.adapt = ADAPT-001 (approved)` ;

the tree is complete (every SR has a `parent`, every SPEC a source); the adversarial review succeeded for every artifact; a pair of TC norms for every statement; the assertions and links cite sections of ADAPT-001. Postcondition: version record `1.0` with the Architect's signature; BR/SR/SPEC have no separate statuses.

5.2 QG-1 — admission of TC implementations. Per §10.3.2, QG-1 applies only to the TC implementation and separates check code admitted to runs from code not yet ready. Preconditions:

`automation.set-version: "1.0"` points to the approved version (V5);
`automation.status` + `location` are valid; static checks and the dry-run pass; the fixing red run is recorded; the mandatory body sections are filled in. Postcondition: the implementation is admitted.

After **QG-0** of version 1.0 and **QG-1** on each TC implementation, TR work can be opened (Phase 6).

Phase 6 — Implementation

*The order is not accidental. The TCs were frozen (`ready`) in phase 5 — **before** the first line of code, and they were not written by the agent that now writes the implementation (standard/09 §9.18). A test born next to the code checks the code, not the requirement. Hence the red-history requirement as well: before implementation the TC is run and **MUST** fail — a test that has never been red proves nothing.*

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

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

6.3 Implementation (fragment):

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

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

```
# acmecorp-login.src/tests/auth/test_registration.py
def test_allowed_domain_succeeds(client, db, mock_ses):
    r = client.post("/auth/register", json={"email": "alice@acmecorp.com",
    "password": "ValidPass123!"})
    assert r.status_code == 201
    assert "user_id" in r.json()
    user = db.query(User).filter_by(email="alice@acmecorp.com").one()
    assert user.verified is False
    mock_ses.send_email.assert_called_once_with(template_id="verification-email",
```

```

to_email="alice@acmecorp.com")

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

```

6.4 Substrate-side validation hook:

```

[hook] Checking links for TR-101: parent.id SR-01 (approved); implements-spec
[SPEC-API-01, SPEC-SEC-01].
[hook] Negative TCs: SR-01.verified-by includes TC-002, TC-004 (negative).
✓ Change allowed.

```

Phase 7 — QG-2 (verification gate)

7.1 CI runs the TCs. `pytest acmecorp-login.src/tests/auth/test_registration.py` → 4 TC PASSED → the Bot updates `last-run.result = pass`, `last-run.set-version = 1.0` in the TC files. Each TC's run history shows the "red → green" transition: the pinning run before implementation was red — the condition under which a TC counts as evidence at QG-2 (§9.18.2).

7.2 Spot-check (standard/09 §9.14). Once per sprint the Engineer manually runs 5 passing TCs selected by machine signals (priority — TCs with no red history, then mutant survivors, then the `implementation-originated` class; the remainder by random top-up), and checks the actual result against the SR. Selected: TC-001, TC-008, TC-012, TC-019, TC-024 → 5/5 match.

7.3 Verification entry of version 1.0. QG-2 preconditions: all TCs of the version passed with `last-run.set-version = 1.0`; both TCs of every pair; red history; the spot-check passed. Postcondition: the runner appends `verification` to the version record 1.0 (§10.5.4) — SR-01 is verified as part of the pair (version 1.0, build), it has no status of its own; the coverage index is updated.

Phase 8 — Delta-TZ

8.1 The client, a week later:

```

# TZ-2026-051 – Addendum to TZ-2026-042
Base: TZ-2026-042

## §2 (change) FR-001 (extension)

```

Additionally allow `@subsidiary.acmecorp.com` (the subsidiary company). The whitelist is extended to 2 domains.

8.2 Delta-ADAPT + ACTZ-002. Operation `adapt-from-tz (delta)` : input — TZ-2026-051 + parent ADAPT-001; output — draft ADAPT-001-delta-1 + delta Forward + backward findings (e.g. B-008 scope). Finding B-008 ("is `@sub.subsidiary.acmecorp.com` to be treated as allowed?") is put to the client as the protocol `ACTZ-002` — "TZ Clarification Protocol No. 2"; once signed, B-008 receives `decided-in: ACTZ-002 §1`, and the delta-ADAPT is approved by the Architect's signature. The effective TZ becomes: `TZ-2026-042 + TZ-2026-051 + ACTZ-001 + ACTZ-002`.

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

Affected:

- BR-01 (scope extension)
- SR-01: changed in the draft → enters version 1.1
- TC-001..004: norms changed → implementations become stale (`automation.set-version 1.0 ≠ 1.1`); +2 new TC (subsidiary domain)
- TR-115: new implementation task
- SPEC-SEC-01: allowed-domains extended

8.4 Apply delta. The Architect opens the changes with the marker `[delta:TZ-2026-051]`. The AI updates SR-01 (extends the whitelist) and generates 2 new TCs. Implementation in TR-115. CI runs the TCs, the bot updates last-run. The draft is approved as version 1.1 (QG-0 of the set); the implementations are re-created against the new norm (`automation.set-version: "1.1"`). After the spot-check — the verification entry of version 1.1.

Note (simple delta). If the adversarial reviewer returns a "no findings, no clarifications" verdict (§7.4.1.2), **no delta-ADAPT is created** — BR/SR/SPEC get `source.tz-section` directly with a fixed `adversarial-review-ref`. A trivial change (a field rename) produces neither an interpretation nor a protocol: there is nothing to ask the client about.

Phase 8a — Three inputs of a set change

A delta-TZ is not the only way a description set gets a new version. A change arrives through three inputs; they differ by the moment and by what is already known, not by cost: each gives a minor version at once ([standard/10 §10.5.1](#)), and tasks grow from the changed set, not from the text of the input.

Input	When	What is known	Provenance in BR / SR / SPEC	What it produces
Delta-TZ (phase 8)	the client changed the promise	the new contract text	<code>source.tz-section</code> to the delta; with findings — ADAPT / ACTZ	a set version; rework with annulment of verification
Concept for a	an engineer sees an improvement	the target structure:	<code>source.adversarial-review-ref</code> + a reference to	a set version (minor)

change	before there is a task — before the task's QG-0	how it should be	the accepted concept in the version record	
Backward finding / problem	during a task or after the fact: a failed TC, a defect, a client remark	only "what is wrong"	depends on the outcome of the analysis (table below)	a diagnosis → an implementation defect or a set version

A concept for a change describes the target structure — without comparisons with the current one, without variants and without open questions; it does not enter the set and is not signed by the client. A concept accepted by the Architect **is entered into the set** and gives a minor; from then on it is version history, not a source of truth. A deferred implementation is a queue entry without a set version of its own. A verbal instruction or a wish is not an input: it is written up as a concept. For decisions verified by the behaviour of a screen rather than by text, the concept is complemented or replaced by a clickable prototype on notional data — only what is entered from it into a SPEC-UI enters the set.

Outcomes of a problem analysis. The analysis establishes who is wrong, and only then does a defect appear. Five outcomes and their place in RENAR:

Outcome	Who is wrong	In RENAR	The set	The contract
C — code	the implementation	a source-of-truth drift (standard/04 §4.11 , class 4.11.3) → a TR against the current version	untouched	untouched
T — test	the TC implementation (the norm is right)	the TC implementation is re-created against the same norm (standard/09 §9.9); the red history is kept	untouched	untouched
D — description	the set	a backward finding of the internal contour (standard/06 §6.13.2): gap , hidden-assumption , contradiction , terminology , feasibility without client-observable consequences	a minor at once or a queue entry	untouched
P — the promise has to change	the contract	a backward finding with a contractual outcome: scope , contradiction , regulatory or any category with client-observable behaviour → an ACTZ draft, dual signature (standard/07 §7.13)	a version after the ACTZ signature; annulment of verification of the affected TCs	changed
N — something not promised is needed	outside the order	scope (out) → a delta-TZ or a new TZ; does not enter the current set	untouched	new

Screening out — "a requirement outside the implementation scope" — does not become a problem and leaves no record. The boundary the agent does not cross: outcomes **D, P, N are declared only by a person** — the Architect for D, the vendor's authorized person together with the Architect for P / N ([standard/05 §5.4](#)); the agent proposes the diagnosis and prepares the ACTZ draft but neither approves the version nor signs. C and T the agent analyzes and fixes itself: the set is untouched, the fix goes as an ordinary task.

Phase 9 — Acceptance: AT against the effective TZ + QG-4

9.1 Regenerating the ATs. Before the trials, an isolated agent re-derives the acceptance tests — **AT** — from the current revision of the effective TZ ([standard/09 §9.19](#)). Its input is only `TZ-2026-042` + `TZ-2026-051` + `ACTZ-001` + `ACTZ-002`. It has no access to the ADAPT, the BR/SR/SPEC, the TCs, or the code, and its model differs from the primary agent's — otherwise it would reproduce the same interpretation and acceptance would degenerate into a second round of verification.

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

## tz_text
"Additionally allow @subsidiary.acmecorp.com (a subsidiary). The whitelist is
extended to 2 domains."
```

A verbatim quote of the contract clause next to the verification steps is a mandatory part of the AT body: at the trials it is the clause of the contract itself that is presented, not a paraphrase of it.

9.2 The release gate. The product is not submitted for delivery until every AT is in status `passing` (§10.4.3). The run gives 11/12 green: `AT-07` ("access recovery via the administrator") is red while every TC is green. That is the most valuable row of the routing table (§9.19.5): AT X / TC ✓ — an **interpretation** error, not a code error. The analysis confirms it: the ADAPT reduced recovery to a password reset, whereas the TZ requires a ticket to IT support. The fix goes into the ADAPT (errata) and the derived SR/TC, after which AT-07 turns green. The `ACCEPTANCE.md` report — AT coverage across the sections of the effective TZ — is refreshed automatically.

9.3 QG-4 (optional) — the business outcome. Four weeks after release the business effect is measured. The gate does not replace contractual acceptance and is not mixed with it: the goal can be met while the contract is not, and vice versa.

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

9.4 Sign-off. The client accepts the result: the BR of the set version is accepted (QG-4), the mark is in the version record. Archive: `ACCEPTANCE.md` + `QG-4-REPORT-v1.0.md` + lessons learned `lessons/2026-Q2.md` .

Final artifacts

```
acmecorp-requirements/  
├─ adapt/                    ADAPT-001-main.md (frozen) + ADAPT-001-delta-1.md  
(frozen)  
├─ actz/                    ACTZ-001.md + ACTZ-002.md (signed, immutable)  
├─ br/                      BR-01 + BR-02 (status: accepted)  
├─ sr/                      SR-01 (v1.1, verified) + SR-02..SR-07 (v1.0, verified)  
├─ specs/                  SPEC-UI-01 + SPEC-API-01 + SPEC-DATA-01 + SPEC-SEC-01  
(verified; SPEC-SEC-01 v1.1)  
├─ tests/                  TC-001..TC-028 (28 TC, 100% passing)  
├─ at/                     AT-01..AT-12 (regenerated before the trials, 100%  
passing)  
├─ tz/                     TZ-2026-042.md + TZ-2026-051.md (delta, immutable)  
├─ elicitation/            # Phase 0 artifacts  
├─ lessons/2026-Q2.md      # Phase 9 lessons  
├─ ACCEPTANCE.md          # AT coverage across the effective TZ  
└─ QG-4-REPORT-v1.0.md    # business-outcome report
```

The effective TZ here is not a file but a computed sum: `TZ-2026-042` + `TZ-2026-051` + `ACTZ-001` + `ACTZ-002` . That is exactly what was fed to the AT generator.

Project metrics

Metric	Value
RDLT (TZ signed → all SR verified)	11 days
Coverage Velocity	100% over 2 sprints
Hallucination Rate (detected)	0%
Adversarial findings found (cycle 1)	4 high + 2 medium
Test-spec drift on the delta-TZ	0%
Signed ACTZs	2 (TZ Clarification Protocols No. 1 and No. 2)
ATs at the trials	12 (1 red → an interpretation error, fixed)

Acceptance disputes	0 (1 finding, resolved before sign-off)
Cost per BR	\$0.46 (gen) + \$0.18 (critic) = \$0.64
Total AI cost	~\$8.50
BRs accepted	2/2
Days to accept	35

What this example shows

1. **Transparency** — every artifact has provenance, every transition is a gate with explicit conditions.
2. **Speed** — decomposing an approved ADAPT — tens of seconds + 2 adversarial cycles.
3. **Traceability** — from a line in the TZ to a passing TC in a handful of substrate query operations.
4. **Delta-TZ** — the affected SR/SPEC/TC/TR are computed automatically.
5. **Two loops instead of one** — the client signs the ACTZ (obligations), the Architect signs the ADAPT (interpretation). The argument "is this a clarification or already a change?" never arises: it is enough to ask whether the decision was put to the client.
6. **Acceptance against the contract** — the ATs, derived by an isolated agent from the effective TZ, caught an interpretation error that all 28 green TCs missed by construction.
7. **Closing the loop** — QG-4 ties the result to business metrics (KPI achievement).
8. **AI-nativeness** — the critic and the generator are different models (isolation); the spot-check finds discrepancies that an automated run alone may miss.

What's next

- [02-transition-guide.md](#) — transitioning from a legacy approach.
- [03-tool-guide-git.md](#) — git as a substrate.
- [04-document-store-substrate.md](#) — a document-oriented substrate.
- [05-safe-comparison.md](#) — comparison with SAFe / BABOK / ISO 29148.
- [06-compliance.md](#) — compliance mapping (GDPR / FZ-152 / AI Act).
- [07-failure-modes.md](#) — failure modes.

RENAR Walkthrough 1.0 — [renar.tech](#)

02. Transitioning to RENAR

Teams rarely start from a blank page. This chapter is about how to move an existing project from the manual "TZ → code" loop to RENAR gradually, step by step, without halting development. Each level delivers tangible value; a team chooses which **RENAR-N** to reach based on real value, not on a race for "full conformance" to claims on paper.

Prerequisites: RENAR Core, standard/11-maturity-model (closed list of levels RENAR-1..RENAR-5), guide/07-failure-modes. Already on early RENAR with legacy types (**INT-TC** , **AIC** , ...) — see 10-migration-v1.

1. Assessment: where the team is now

Before migrating, assess the current state. A "pre-RENAR" checklist:

Signal	Pre-RENAR	RENAR-1 minimum
The TZ exists as an artifact	In chat / Google Doc / Notion	Fixed in the substrate as a file
Someone maintains the requirements	Only in a tracker (Jira / Linear)	In the substrate as BR / SR / ADAPT
Where a requirement came from	From memory / we re-ask	Anyone can find the source in the substrate
Requirement changes	Verbally at standup	Through an explicit change-set (delta-TZ)
Tests	In code, no link to requirements	TC as artifacts, or at least in code with an SR-ID mention

If 3+ rows fall in the "Pre-RENAR" column, the team is **below RENAR-1**. This is a normal starting point for most projects.

1.1 Readiness signals

A team is ready to migrate if:

- It hurts that requirements get lost between chats / tickets / documents.
- Disputed acceptances happen regularly — "this isn't what we asked for".
- Onboarding a new engineer takes months to reconstruct context.
- AI is used to generate requirements / code, but there is no systematic verification.

If none of this hurts, RENAR may be premature optimization. See §8 "when it is not needed".

2. Stage 1 — entering RENAR-1 (Ad-hoc)

Level goal: requirements live in the substrate, not in chat; every TZ has gone through adversarial review with an issued AR, and an ADAPT is created on a "findings present" verdict.

Typical duration: 1-2 weeks.

2.1 What gets added

1. An artifact **substrate** is chosen and set up: a `<project>.req/` directory in the repository, a document-store workspace, or another substrate — RENAR is not tied to a specific implementation; see capabilities **V1–V6** (glossary §2.7).
2. The existing TZ is moved into this environment as an artifact not subject to arbitrary edits (with date and signatures).
3. Every TZ goes through adversarial review; its outcome is issued as an **AR** — an adversarial-review record (standard/07 §7.4.6). On a "findings present" verdict an **ADAPT** is created — a bridge artifact: a forward-interpretation section ("how we understood it") and a backward-findings section. On a "no findings" verdict no ADAPT is created, and BR / SR / SPEC reference the TZ directly through `source.tz-section` + `source.adversarial-review-ref` (§7.4.1).
4. New requirements are recorded as BR / SR files in the substrate, not only in the tracker.

2.2 What is NOT required at this stage

- Standardized frontmatter (at minimum — `id` + `title`).
- Lifecycle statuses (`draft` / `approved` /...) — artifacts MAY live without explicit transitions.
- TC as separate artifacts.
- Substrate hooks.
- Substrate-native COVERAGE.

2.3 Common blockers

- **"We already have tickets in Jira"** — leave them. RENAR-1 does not require migration; the tracker and the substrate can coexist. The key point is that the substrate is now the Source of Truth for **new** requirements.
- **"ADAPT is extra work"** — an ADAPT takes 1-2 hours per TZ. It pays off at the very first disputed acceptance.
- **"Where do we store it?"** — any substrate that satisfies **V1–V6**. See [guide/03-tool-guide-git](#) or [guide/04-document-store-substrate](#).

2.4 When to move to RENAR-2

When **new requirements** have stopped going into chats and started appearing in the substrate reliably. This takes 4-6 sprints of habit-building.

3. Stage 2 — moving to RENAR-2 (Documented)

Level goal: structure and frontmatter; delta-TZ as an explicit change-set.

Typical duration: 2-4 weeks after RENAR-1.

3.1 What gets added

1. Every BR / SR gets frontmatter with mandatory fields (see [reference/02-schemas](#)).
2. Folders are structured: `br/` , `sr/` , `adapt/` , `actz/` , `dpia/` , ...
3. Requirement changes — only through a delta-TZ (a new immutable artifact), not through direct editing.
4. The TZ is fixed as immutable (changes only through a delta-TZ).
5. Clarifications obtained from the client without a new TZ are drawn up as an **ACTZ** — a TZ clarification protocol with a dual signature ([standard/07 §7.13](#)) — not as a message in a chat.

What this step really changes. Before RENAR the team has one contractual document — the TZ — and everything discussed after it was signed lives in correspondence. RENAR separates two things that used to be conflated:

What	Where	Who signs
A decision put to the client and accepted by them	ACTZ ("TZ Clarification Protocol No. N")	The client and the contractor
The engineer's reading of the TZ — terms, elaborated scenarios, findings	ADAPT	The Architect alone (§7.5)

The client does not sign the ADAPT: that used to be a fiction — a signature under an engineering document the client never read. The practical upshot is the **effective TZ** (§7.14): the initial TZ plus every signed ACTZ. It — not the correspondence and not the ADAPT — becomes the benchmark for delivery and acceptance.

3.2 Template: legalizing existing requirements

Old requirements (already in the substrate since RENAR-1) — run a review and stamp frontmatter "as-is" without revising the content. This is **legalization**, not **rewrite**:

```
---
id: BR-12
title: "Employee registration via corporate email"
# no status – the BR belongs to the approved set version (already running in prod)
created-at: "2025-12-01" # retroactively
priority: must           # retroactive assessment
legacy: true             # marker: the requirement did not go through ADAPT from scratch
---
```

The `legacy: true` field is optional but RECOMMENDED — it distinguishes "historical" requirements from new ones that went through the full RENAR pipeline from the start.

3.3 Common blockers

- **"frontmatter for 100 requirements is a month"** — yes. Do it in the background, 10-15 requirements / week; new requirements get frontmatter immediately.

- **"Delta-TZ slows us down"** — in the first weeks, yes. After 2-3 iterations it usually becomes faster than "let's just change it".
- **"We don't know what priority to set retroactively"** — leave `priority: should` for all legacy; an explicit `must` is set only when confirmed with a Stakeholder.

3.4 When to move to RENAR-3

When frontmatter is valid for 80%+ of artifacts and the team has gotten used to delta-TZ.

4. Stage 3 — moving to RENAR-3 (Tracked)

Level goal: automatic validation + lifecycle enforcement + TC coverage for priority=must.

Typical duration: 4-8 weeks after RENAR-2.

4.1 What gets added

1. Substrate hooks validate frontmatter against the schema on every change. Invalid ones — integration is blocked.
2. The lifecycle is run for real: description set versions are released (`draft` → `approved N.M` → `superseded`); tasks and ADAPTs have their own closed states (`standard/10 §10.5`).
3. TC are created for all priority=must BR / SR / SPEC.
4. A substrate-native COVERAGE report is auto-generated on every set version release and every verification entry.
5. Reference-validation hook: creating a BR / SR with a link to an ADAPT in a status below `approved` — blocked.
6. The implementation references BR / SR / SPEC with a pinned `verifies[].version` .

4.2 Template: backfilling TC

For all priority=must requirements without a TC:

1. Sort by "frequency of mention in incident reports" — where a TC delivers the most value.
2. Cover 3-5 requirements / sprint, without trying to backfill everything at once.
3. `TC` are created as artifacts in your substrate with a `verified-by` link.

4.3 Common blockers

- **"Old requirements stubbornly resist the schema"** — leave them in a legacy folder; new ones are schema-valid right away. The substrate hook applies only to new / modified artifacts.
- **"Hooks slow down the PR cycle"** — measure: if > 30s — optimize the hooks (parallelization, caching); do not "turn them off".
- **"The team bypasses hooks via `--no-verify`"** — this is an [organizational failure pattern](#); the root cause is that the hooks are too slow / noisy. Fix them, do not forbid.

4.4 When to move to RENAR-4

When COVERAGE for priority=must = 100%, frontmatter is valid everywhere, and the lifecycle actually works.

5. Stage 4 — moving to RENAR-4 (Verified)

Level goal: all statements of the approved set version have successfully passing test cases (`TC`); the verification entry of the version under the `QG-2` gate (`Verification Gate`) is enforced by the substrate; positive / negative pairing is observed; an `ai-provenance` block is set for AI-sourced material.

Typical duration: 6-12 weeks after RENAR-3.

5.1 What gets added

1. 100% of BR / SR / SPEC of the approved set version have a `verified-by` link to ≥ 1 TC norm of the same version.
2. Pos/neg pairing for every normative clause.
3. The `QG-2` gate (`Verification Gate`) is blocked by substrate-built-in checks: the verification entry of a version — only when all `TC` s pass with `last-run.set-version` equal to the version.
4. All TC are automated or explicitly `manual-pending` with a deadline.
5. For `tc-type: ux` — VLM-judge isolation.
6. For `tc-type: eval` — the judge model $\neq$ the implementation model.
7. `ai-provenance` for AI-generated artifacts — in the composition of the canonical `standard/04 §4.10.1 : generated-by` (including the model version), `generated-at`, `prompt-template`, `context-tokens`, `output-tokens`, `human-edits`.
8. Source citation — every normative clause has a pointer to a source in the TZ or ADAPT.
9. Reconciliation is run by the substrate at least once a week.
10. Spot-check 5 random passing TC once a sprint.
11. Test-authorship isolation: the TC norm is approved in a version and the implementation is admitted (QG-1) before the behaviour is implemented, and it is not written by the agent that writes the code (`standard/09 §9.18`).
12. Red history: the pinning run of a TC before implementation MUST be red. A test that is green before the code exists is a signal to investigate, not a success.

Acceptance against the contract. At this same stage a second layer of checking appears — **AT** (`standard/09 §9.19`). TCs prove conformance to the interpretation; ATs prove conformance to the effective TZ. They are generated by an isolated agent whose input is **only** the effective TZ (no ADAPT, SR, TC, or code) and whose model differs from the primary agent's. ATs are regenerated before every round of trials; until every AT is green, the product is not submitted for delivery (§10.4.3).

What this buys in practice: a team with green TCs and a red AT is looking at an error of **interpretation** of the TZ — the class of defect no TC catches, because the TCs were derived from the very interpretation the code was.

5.2 Template: phased coverage

Reaching 100% verified-by coverage is not a single PR. The approach:

1. First pos-only TC for all priority=must (the benefit — fast feedback).
2. Then neg-TC pairs (the benefit — catching test-fitting drift).
3. Then **tc-type** extensions (ux / eval / contract / security) as needed.

5.3 Common blockers

- **"Judge-model isolation is expensive"** — true. But it is a structural rate limit on [AIR-06 test-fitting drift](#) — it cannot be circumvented without losing verification integrity.
- **"Source citation slows authors down"** — automate it: the AI generator should emit the citation right away; manual authoring is only for legacy backfill.
- **"Reconciliation findings overwhelm the team"** — tunable thresholds; start with conservative defaults and loosen gradually.

5.4 When to move to RENAR-5

When RENAR-4 runs stably for 2-3 quarters, the metrics are stable, and the team is not "burning out" from reconciliation noise.

6. Stage 5 — moving to RENAR-5 (Optimized)

Level goal: adversarial review as a gate; multi-model agreement for priority=must; cost/latency budgets; knowledge graph as primary search; continuous evaluation for AI-critical components.

Typical duration: 12+ weeks after stable RENAR-4.

6.1 What gets added

1. An adversarial critic — a mandatory gate for **draft** → **approved**. The critic is a model different from the generation model.
2. Multi-model agreement for priority=must: the artifact is generated by ≥ 2 models; divergences are flagged and **MUST** be reviewed.
3. Cost / latency budget per artifact; an overrun → automatic decomposition.
4. Knowledge graph as primary search for AI agents.
5. Continuous evaluation for SPEC-AI.
6. Hallucination Rate metric < 1%.
7. Multi-model Disagreement Rate metric is tracked.
8. Feeding template improvements back into the **requirements-library** is standard practice.

6.2 When RENAR-5 is needed

- Regulated industries (fintech, healthcare, high-risk AI systems per the AI Act).
- When the product critically depends on AI-generation quality (generative products).
- When there is a budget for continuous-evaluation infrastructure.

Many projects can stop at RENAR-4 — it is enough for **conformance to a declared RENAR profile**. RENAR-5 is not necessarily "better", but it is almost always **more expensive and stricter**. Choose by need, not by fashion.

7. Migrating legacy requirements

Thousands of existing requirements in Jira / Confluence — how do you pull them in?

7.1 Non-migration strategy

If legacy requirements are not being actively changed — **do not migrate**. RENAR applies only to new requirements and actively changed ones. Legacy stays in its original place as read-only "historical context".

7.2 Selective-migration strategy

For legacy that is actively changing:

1. At the moment of the first change — pull it into the substrate as a BR/SR with a `legacy: true` marker and minimal frontmatter.
2. Apply the change as a delta-TZ.
3. After 1-2 iterations the requirement "matures" — it becomes full-RENAR (no legacy marker, with full frontmatter and TC).

7.3 Bulk-import strategy

When you need to migrate > 100 requirements at once (for example, during an organizational reorganization):

1. Scripted export from the tracker → a frontmatter skeleton (id, title, minimum).
2. All imported requirements — `legacy: true` + `status: approved` (as in prod).
3. Backfill TC and full frontmatter — sprint by sprint, without blocking current work.

8. When RENAR is NOT needed

RENAR is overhead. Do not apply it to:

- **One-off scripts and prototypes** — they never reach production, or they become production only through a rewrite.
- **Very small teams (1-2 people)** — the overhead of managing a substrate may exceed the benefit.
- **Projects with no AI-generation of requirements / tests** — the main value of RENAR (protection against AI-specific failure modes) is lost.
- **Short-lived experiments (≤ 1 sprint)** — you will not have time to recoup the ADAPT setup.

Minimum payback threshold: a project with ≥ 3 requirement iterations, ≥ 2 developers, using AI somewhere in the pipeline (generation of requirements / code / tests / documentation).

9. Migration anti-patterns

What to avoid:

9.1 Big-bang migration

Symptom: The team stops development for 2 sprints to "move to RENAR". After 2 sprints they get burnout and an abandoned migration.

Instead: Hybrid mode. New requirements go straight into RENAR, old ones — as they are touched. Slow, but sustainable.

9.2 Skipping levels

Symptom: "Let's go straight to RENAR-4". The team skips RENAR-2/3, puts in full hooks + ai-provenance + an adversarial critic, but frontmatter is not valid and the lifecycle is chaotic.

Instead: Levels go in order. RENAR-4 without a RENAR-3 base does not work.

9.3 "Perfect frontmatter" paralysis

Symptom: The team spends weeks polishing one piece of frontmatter — "did I pick the right priority ?" The real work stalls.

Instead: frontmatter is "good enough". Mistakes are fixed in a delta-TZ. The point is that something is in the substrate, not that it is buffed to a shine.

9.4 Partial adoption of the substrate

Symptom: Half the requirements are in the substrate, half are still in Jira. Nobody knows where the Source of Truth is.

Instead: The Source of Truth **MUST** exist **right away**. If you cannot move everything — move only the new ones and explicitly declare the substrate the owner of "everything new".

9.5 "Tooling first" approach

Symptom: The team spends half a year writing internal tooling around RENAR (its own validator / its own CI / its own UI) without using the standard itself.

Instead: Practice first, on a minimal substrate (even a flat folder of markdown). Tooling — when it starts to hurt by hand.

Adoption cost and benefit (illustrative)

*This section is **illustrative and non-normative** — it helps estimate orders of magnitude. The concrete numbers depend on the project; calibrate the real model against your own data.*

Adoption cost (what you put in):

- Training the team on [RENAR Core](#) — on the order of half a day to a day per engineer.
- Adversarial review of every TZ, and ADAPT discipline where the review produced findings — additional hours on backward findings before coding starts (recouped by not re-opening the TZ later).
- The substrate already exists (git) — no separate licenses needed; tooling automation is optional and introduced gradually.

Benefit (what you get back):

- Reduced TZ-decomposition time with AI acceleration (order of magnitude — [standard/12 §12.5.1](#): 5–10× at RENAR-4, illustrative).
- Fewer disputes at acceptance: a signed ACTZ fixes the obligation before code, and delivery runs against the effective TZ rather than against someone's memory of the email thread.
- Fewer incidents from negative scenarios — thanks to mandatory pos/neg TC pairing.
- An audit log of "what we delivered under contract" — free from the substrate (V1 / V6).

When it does not pay off: short-lived prototypes, pure product discovery without a contract, teams with no compliance pressure and no external client (see §8 "when RENAR is not needed"). For them, [RENAR Core](#) or nothing is enough.

*A quantitative ROI model (per-project savings order, etc.) currently lives in the research materials; porting a calibrated version into this section is planned in the **phase-8 backlog** for v1.1.*

10. Related documents

- [standard/11-maturity-model](#) — normative definitions of the RENAR-1..RENAR-5 levels (closed list).
- [00-quickstart](#) — a 30-minute sample for a small project.
- [01-walkthrough](#) — a full example on one project.
- [07-failure-modes](#) — what can go wrong during migration; organizational failure patterns §5.
- [reference/02-schemas](#) — frontmatter schemas, mandatory for RENAR-2+.
- [03-tool-guide-git](#) — a substrate-specific guide for git.
- [04-document-store-substrate](#) — an informative overview of a document-oriented store substrate.

11. Decisions fixed for v1.0

- **Legacy backfill bulk-import — bypass is forbidden.** On initial import, artifacts are entered with the status `imported-legacy` (a substrate-specific marker, not part of the normative closed-list lifecycle [§10.5–§10.8](#)) and do not participate in conformance assessment until promoted through the normal QG-0 flow. This preserves the [§1.7.3](#) declared-weaker prohibition: validation is not bypassed, only deferred until the moment the artifact begins to claim conformance.
- **A rollback from RENAR-3 → RENAR-2 is permitted** through a formal downgrade per [§13.8.2](#): a new version of the manifest is issued with a lowered `level`. A recovery plan is **not** mandatory (a downgrade is intentional, not a loss of conformance), but it is RECOMMENDED to record the reason for the downgrade in the audit log.
- **Cross-org migration: each subsystem — its own manifest with its own `level`** ([§13.4](#)). The organization-aggregate "RENAR-N" is informally defined as the minimum of the subsystems' levels; an organizational manifest is **not** standardized by RENAR v1.0.

11.1 Deferred to v1.1 (phase-8 backlog)

- **Migration templates for teams of 50+ engineers.** The guide targets groups of 5–15 people; a larger scale needs field observations. Owners: RENAR standard maintenance and early adopters.

03. Substrate: VCS — git

A concrete implementation of RENAR on git. The `.req` repository structure, submodule pinning between `.req` and `.src`, the PR/MR review workflow, pre-commit hooks for capabilities V1-V6, and the delta-TZ workflow. This guide is informative; the normative content (capability requirements, schemas, lifecycle) lives in `standard/`. For an alternative on a document-oriented store, see [guide/04-document-store-substrate](#).

Prerequisites: [standard/03-substrate-versioning](#) (the normative capabilities V1-V6), [reference/02-schemas](#) (frontmatter schemas).

1. When to choose git as the substrate

Git is the default substrate for:

- Open standards and projects (no dependency on internal infrastructure).
- External clients with no dedicated document-store infrastructure.
- Teams with an established PR workflow.
- Projects where requirements and code share one ecosystem (the same VCS provider).

Git is **not** optimal when:

- You need frequent concurrent edits to a single artifact by several authors without merge conflicts.
- You need built-in full-text search without external tools.
- You need a UI for non-technical stakeholders (PMs, legal) without the git CLI.

In such cases, consider a document-oriented store — [guide/04](#).

2. Two-repository layout

The canonical structure is two linked repositories.

```
<project>/
├── <project>.req/
│   ├── tz/
│   ├── actz/
│   └── signature)
│       ├── adapt/
│       └── ar/
§7.4.6)
│   ├── br/
│   ├── sr/
│   └── specs/
int/, ...)
│   ├── arch/  api/  data/  ui/  ai/  int/  proc/  sec/  ops/  test/  doc/
│   ├── tr/
│   └── tests/
```

← REQUIREMENTS repository (a separate VCS repo)
← TZ-YYYY-NNN.md (immutable after registration)
← ACTZ-NNN.md (TZ clarification protocols, dual
ADAPT-NN.md (bridge artifacts)
AR-NNN.md (adversarial-review records,
BR-NN.md
SR-NN.md
SPEC-* by subfolder (arch/, api/, data/,
TR-NN.md (task requirements)
TC-NN.md (test cases; `TC` are standalone

```

artifacts)
|   |— at/                                ← AT-NN.md (acceptance tests from the effective
TZ)
|   |— dpia/                             ← DPIA-NN.md (optional, for regulated projects)
|   |— library/                          ← templates, patterns
|   |— docs/                             ← AI-generated documentation
|   |— COVERAGE.md                      ← auto-generated (`[coverage]` commits)
|   |— REQUIREMENTS.md                  ← auto-generated index
|   |— TEST-PLAN.md                     ← auto-generated
|   |— <project>.src/                   ← IMPLEMENTATION repository
|   |   |— src/                         ← code
|   |   |— tests/                       ← TC implementations (addressed by
`automation.location`)
|   |   |— requirements/                 ← submodule → <project>.req @ <commit>
|   |   |— .gitmodules
|   |   |— README.md

```

In `<project>.req/.gitattributes` , for bot-generated artifacts:

```

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

```

This excludes them from code statistics and substantially shrinks the PR diff.

3. Capability mapping V1-V6 onto git

Capability (standard/03 §3.3)	Norm	Git mechanism
V1 — immutable history	Past artifact states cannot be rewritten retroactively	Immutable commit history; protected branch + force-push ban on <code>main</code> ; the <code>id:</code> in frontmatter is stable
V2 — atomic change unit	A change applies as a whole or not at all	An atomic commit / squash-merge PR as a single unit; a delta-ADAPT = one PR
V3 — diff & review	A proposed change is reviewed before approval	<code>git diff</code> + PR/MR review with a mandatory approve before merge
V4 — branching / change-set	A draft is kept separate from the approved truth	The set-draft branch against <code>main</code> (the approved version); a PR is a change-set, merging = release of version <code>N.M</code>
V5 — cross-substrate version pin	The implementation references a specific set version	Submodule pin between <code>.req</code> and <code>.src</code> ; the set version tag <code>renar-vN.M = set-version</code> ; <code>automation.set-version</code> in TC
V6 — author & timestamp	Every change unit records its author and time	Commit metadata: author + date; for an AI agent — <code>ai-provenance</code>

RENAR checks on git (schema validation, status-transition control, coverage reports, link integrity, reconciliation / drift detection) are **enforcement** mechanisms layered on top of the capabilities, not V1–V6 themselves. Their split across pre-commit / CI is §3.1 and §8 of this guide; the normative drift classes are standard/04 §4.11.

State of these scenarios. The mechanisms described below for `git` are the **v1.0 target picture**. In practice only `scripts/validate-frontmatter.js` is ready; the rest (`validate-lifecycle` , `validate-references` , `generate-coverage` , `detect-drift` , and others) are in the **phase-8 backlog** (section §8 of this guide). Until the scripts exist, the listed capabilities are upheld manually and through code review; automatic enforcement of RENAR-3+ levels "out of the box" on `git` is not yet achievable. The same caveat applies to the document store (guide/04).

4. Submodule pinning

`<project>.src` pins a **specific commit** of `<project>.req` through a git submodule.

4.1 How it works

- In `<project>.src` , the `requirements/` directory is a submodule on `<project>.req` .
- At build / CI time the code knows: "I implement the requirements as of commit `abc1234` ."
- A developer working a task opens `requirements/sr/SR-05.md` via an ordinary `cat` or the IDE — it is a file in the worktree.

4.2 Bump pattern

When the requirements are updated:

1. A PR into `<project>.req` with the requirement changes → review → merge.
2. A **separate** PR into `<project>.src` that **only** moves the submodule pointer:

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

3. This PR makes it explicit: "the requirements were updated to commit X."

4.3 Why submodule, not subtree / monorepo

- **Provenance:** for any code commit, the exact requirement version it implemented is known.
- **Review isolation:** requirement review (in `.req`) and code review (in `.src`) do not get mixed.
- **delta-TZ atomicity:** a delta is an atomic PR into `.req` plus a subsequent submodule bump in `.src` . There is no "partially applied delta."

- **Document-store compatibility:** when migrating to a document-oriented store, submodule pinning becomes revision-token pinning — the concept is the same.

Alternatives (subtree, monorepo): consider them only if a submodule does not work for reasons specific to your VCS provider; in all other cases submodule is the recommendation.

5. PR/MR review workflow

Two review levels, split across the repositories.

5.1 Review in `<project>.req`

Reviewer focus:

- frontmatter is schema-valid.
- The citation to the TZ / ADAPT is present and valid.
- The `parent` / `verified-by` / `constrained-by` links exist.
- The lifecycle transition is legitimate.
- A source citation is present in the body for every normative statement (at RENAR-4+).
- The adversarial AI-review prompt passed (at RENAR-5).

Approval = QG-0 ([standard/10 §10.3.1](#)).

5.2 Review in `<project>.src`

Reviewer focus:

- The submodule pointer matches a merged commit in `.req`.
- The implementation references the approved set version through `automation.set-version` (tag `renar-vN.M`).
- New / changed TC match the contract in `.req`.
- No changes to TC pass/fail criteria without a `[test-spec-change]` tag.

Merging the draft branch into `main` = QG-0 of the set (release of version `N.M`, tag `renar-vN.M`); **QG-2** ([standard/10 §10.3.3](#)) — the verification entry of the version after all TCs pass on it.

5.3 Forbidden anti-patterns

- **A PR spanning both `.req` and `.src`** — it breaks review isolation; the substrate hook forbids it.
 - **Changing the submodule pointer without a merged commit in `.req`** — the pointer references an untracked commit; CI blocks it.
 - **`[test-spec-change]` without a separate approval** — a TC pass/fail criterion changed together with a code fix; CI blocks the merge.
-

6. Pre-commit and pre-merge hooks

The minimal substrate hook set for RENAR-3+ on git.

6.1 Pre-commit (in `<project>.req`)

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

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

# Capability V5: reference integrity (fast check of changed files only)
node scripts/validate-references.js --changed-only $(git diff --cached --name-only)
```

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

The full checks that are too slow for pre-commit:

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

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

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

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

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

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

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

7. TZ workflow on git

7.1 Forward workflow (project from scratch)

Initial creation of the requirements axis from a new TZ:

1. **branch in .req** : `git checkout -b init/TZ-2026-001` in `<project>.req` .
2. **TZ registration**: `tz/TZ-2026-001.md` (immutable after registration; record the client signature and date).
3. **Adversarial review of the TZ and — on findings — the ADAPT**: the review is always mandatory and its verdict is recorded as `ar/AR-001.md` (§7.4.1.2). On a "findings present" verdict, `adapt/ADAPT-001.md` is created — Forward (an interpretation per § of the TZ) + Backward (findings), `standard/07`; the ADAPT is an internal document and is not put to the client. On a "no findings" verdict no ADAPT is created, and BR / SR / SPEC carry `source.tz-section` + `source.adversarial-review-ref: AR-001` (§7.4.1.1).
4. **ACTZ**: findings that require a decision from the client are put forward as the protocol `actz/ACTZ-001.md` — "TZ Clarification Protocol No. 1" (§7.13). `status: draft` → `sent` → `signed` ; the client's and the contractor's signatures are recorded with author + timestamp (V6). Once signed, every finding receives `decided-in: ACTZ-001 §M` .
5. **Architect's signature** → **QG-ADAPT-approve**: ADAPT moves to `approved` (the Architect's signature alone, §7.5), `open-questions-count == 0` .
6. **Decomposition**: the AI agent derives BR from the approved ADAPT, then SR (`source.adapt`) and SPEC (`source.adapt` ; SR references them through `constrained-by[]` , §8.6.1), plus paired pos/neg TC.
7. **QG-0 approval** of each artifact (`draft` → `approved`); a CI dry-run of the new TC.
8. **PR into .req** → **merge**; the bot regenerates REQUIREMENTS.md / COVERAGE.md / TEST-PLAN.md.
9. **Set up .src** : add the submodule `requirements/` → `<project>.req @ <commit>` , create TR, implement, QG-2.
10. **AT before the trials**: an isolated agent (a different model, with no access to `.src` or to the internal artifacts of `.req`) derives `at/AT-NN.md` from the effective TZ — `tz/` plus the signed `actz/` (§7.14). The run refreshes `ACCEPTANCE.md` ; the release gate requires every AT to be green (§10.4.3).

7.2 Delta-TZ workflow

The full sequence for applying a new delta-TZ.

1. **branch in .req** : `git checkout -b change/TZ-2026-042` in `<project>.req` .

2. **delta-TZ creation, adversarial review, and — on findings — the delta-ADAPT:** `tz/TZ-2026-042-delta.md` (the delta text); the adversarial reviewer issues a verdict, recorded as `ar/AR-NNN.md`. On a "findings present" verdict, `adapt/ADAPT-NNN-delta.md` is created (forward interpretation + backward findings, [standard/07 §7.6](#)); on a "no findings" verdict no delta-ADAPT is created and the affected BR / SR / SPEC reference `source.tz-section: TZ-2026-042-delta` directly ([§7.6.1](#)). The client's decisions on the delta-TZ findings are recorded in a new `actz/ACTZ-NNN.md` (dual signature); the delta-ADAPT is approved by the Architect's signature before the affected requirements are modified. The effective TZ is recomputed — and the ATs are regenerated from it.
3. **Impact analysis:** the AI agent finds the affected BR / SR / SPEC / TC; marks the TC as `obsolete-pending`.
4. **Update artifacts:** the AI agent updates / creates BR / SR / SPEC and paired TC (pos+neg) on the same branch.
5. **Adversarial critic review:** at RENAR-5 — mandatory (a different AI model).
6. **CI dry-run:** the new TC must run without infrastructure errors.
7. **Finalize:** merge the draft into `main` = set version `N.M+1` (tag `renar-vN.M+1`, version record), regenerate `REQUIREMENTS.md` / `COVERAGE.md` / `TEST-PLAN.md` (bot).
8. **PR into `.req` → QG-0 approval → merge.**
9. **branch in `.src`:** `git checkout -b change/TZ-2026-042 in <project>.src`.
10. **Bump submodule:** `cd requirements && git pull && cd ..`.
11. **Create TR / tasks:** for new / changed SR (via `/task` or substrate-specific tooling).
12. **PR into `.src` "bump requirements + tests + new tasks" → review → merge.**
13. **Development:** pick up a TR → implement → CI → the bot fills `last-run` in the TC.
14. **QG-2:** on green TCs on the version — the runner appends the verification entry to the version record; the requirement has no separate status.

Every step except (1) and (9) is automated natively for the substrate through scripts or CI.

8. Migration notes: scripts/ modernization

The existing [scripts/](#) are bash + node helpers from an early era of the project. Modernization status as of v1.0:

-  **Legacy terms cleaned up.** `req-branch.sh`, `req-finalize.sh`, `req-ai-instructions.md`, `req-use-template.sh` no longer use the deprecated `INT-SR`, `AIC`, `UIC`, `tech-specs`, `ai-concepts`, `ui-concepts`. The closed list of current types is [standard/04 §4.4](#) (BR/SR/TR + 11 SPEC). Residual mentions in [reference/01-glossary.md](#) and [reference/02-schemas.md](#) are legacy mappings for projects migrating from earlier versions.
-  **Schema validator created.** `scripts/validate-frontmatter.js` — a Node ES module that checks the frontmatter of every `.md` in `standard/`, `guide/`, `reference/`, `core/`: required fields `title` / `order` / `lang` $\in \{\text{ru}, \text{en}\}$. Run: `node scripts/validate-frontmatter.js [--quiet]`; exit 0 if all are valid, exit 1 on the first error. The schema source is [reference/02-schemas](#).
-  **ADAPT in the forward workflow.** Initial creation (TZ → adversarial review → an ADAPT on findings → BR) is documented in §7.1 of this guide; the delta workflow with the delta-ADAPT is in

§7.2, per [standard/07 §7.6](#).

- 🕒 **Pre-commit hooks** (§6.1) are still partly scattered across `req-finalize.sh`. The target state is to factor them into separate `scripts/validate-*.js`; `validate-frontmatter.js` is the first such module.

This chapter describes the **target script set** at the v1.0 release; lines without a  mark are work for **phase 8** (continuation of the backlog).

9. Common operations

Shortcuts for frequent operations.

Operation	Command
Find an SR by ID	<code>grep -r "^id: SR-12" sr/</code>
Find the TC verifying SR-12	<code>grep -r1 "id: SR-12" tests/</code>
Find an orphan SR (no TC)	<code>node scripts/find-orphans.js sr</code>
Create a new SR from a template	<code>cp library/templates/sr.md sr/SR-NN.md && \$EDITOR sr/SR-NN.md</code>
Diff frontmatter over a period	<code>git log --all --since=... -- sr/</code>
Current set version	<code>git describe --tags --match 'renar-v*'</code>
List stale TC (last-run < current version)	<code>node scripts/list-stale-tc.js</code>

10. CI/CD integration patterns

10.1 Bot-commit conventions

Auto-generated artifacts are committed by substrate hooks with conventional commit-message tags for machine parsing:

Tag	When	What is committed
<code>[coverage]</code>	Post-merge in <code>.req</code>	Regeneration of COVERAGE.md / REQUIREMENTS.md / TEST-PLAN.md
<code>[baseline-update]</code>	After approval of a PR that changes a baseline	Regeneration of PNG baselines for SPEC-UI
<code>[bump-req]</code>	Submodule bump in <code>.src</code>	Only the submodule pointer + minimal metadata
<code>[reconcile]</code>	The reconciliation hook finds drift	Auto-fix of obvious mismatches; flag for the rest

[test-spec-change]	A TC pass/fail criterion changed	Manual; requires a separate approval from a reviewer
--------------------	----------------------------------	--

The substrate hook parses the commit message and applies different validation rules depending on the tag. [coverage] / [bump-req] / [reconcile] commits come from the bot user; [baseline-update] / [test-spec-change] come from a human + bot signature.

10.2 Bot-user setup

- A separate bot account (e.g. `renar-bot@<org>`) with **write** permission in `<project>.req` and `<project>.src`.
- Bot commits are signed (GPG / SSH commit signature).
- The bot user **cannot** approve a PR (separation of duties — approval is always human).

10.3 branch protection

In both repositories:

- `main` (or `master`) is protected. A push requires a merged PR.
- Required status checks: schema validation (V2), reference integrity (V5), lifecycle validation (V3).
- Reviewers required: ≥ 1 for most; ≥ 2 for priority=must BR / SR / SPEC.

11. Cross-references

- [standard/03-substrate-versioning](#) — the normative substrate requirements (capabilities V1-V6).
- [reference/02-schemas](#) — frontmatter schemas for the validation hooks.
- [02-transition-guide](#) — where this guide fits into the path from pre-RENAR to RENAR-N.
- [04-document-store-substrate](#) — an informative overview of the document-oriented store (the git alternative).
- [07-failure-modes](#) — what can go wrong on git as the substrate (the hook-bypass scheme, etc.).
- [standard/10-lifecycle-qg](#) — the normative lifecycle transitions that the validate-lifecycle hook must enforce.

12. Open questions

- Should the minimal hook implementations be standardized as a **single** specification description that both VCS and the document store rely on?
- Submodule vs subtree: which conditions make subtree an acceptable alternative? The guide is currently unambiguously in favor of submodule.
- [coverage] bot user: best practice for signing / permissions in multi-org projects?
- Drift-detection cadence: weekly is a good default, but what about high-velocity teams (> 50 PR/week)?

04. Substrate: document-oriented store (overview)

Informative appendix. The normative V1–V6 capabilities are in [standard/03](#). A practical example on a distributed VCS ([git](#)) is in [guide/03](#).

A **document-oriented store** is a substrate where RENAR artifacts are stored as versioned documents: a stable identifier, a chain of revisions (revision token), atomic update, and an API gateway for lifecycle transitions and signatures.

RENAR is formally **not tied** to the document-store class of systems — only the capabilities (**V1–V6**) are normatively prescribed ([§3.3](#)).

1. When to choose a document-oriented store

A good fit if:

- you need a centralized enterprise requirements store across several projects;
- non-technical stakeholders work through a web UI rather than through a VCS;
- federation of cross-project links and search is a **key** product requirement;

A poor fit if:

- the team is already standardized on a git workflow with PR/MR review;
- there are no resources to maintain a document-store server + search index + API gateway;
- you need **fully-fledged** test cases (`TC`) as objects in the same environment without separately configuring custom document types (see [§9](#) — the presence of `TC` is required for declared RENAR conformance);

2. Mapping the V1–V6 capabilities (generalized)

Capability	Typical document-store mechanism
V1 — immutable history	A chain of immutable document revisions + a stable <code>_id</code>
V2 — atomic unit of change	A single whole document-version increment per step
V3 — comparison and review (diff)	An approval flow through the API and the UI; interception of direct status edits
V4 — branching and merging	Draft documents or conflict branches (depending on product capabilities)
V5 — version pinning	A field referencing the revision in linked artifacts
V6 — author and timestamp	Document fields <code>author</code> + <code>updated_at</code> per revision

A comparison with a distributed VCS is in §3.4 (an illustrative table; not a normative contract).

3. Migration VCS ↔ document store

Migration is possible if both substrate implementations preserve:

- the canonical artifact identifiers (BR / SR / SPEC / ADAPT);
- the traceability links;
- the lifecycle states from the closed list in §10.

The migration procedure is a project-specific runbook; the normative minimum is a check of V1–V6 before cutover (§3.8).

4. See also

- [03-tool-guide-git.md](#) — the substrate-specific guide for a distributed VCS (git)
 - [03-substrate-versioning.md](#) — the normative V1–V6
 - [02-schemas.md](#) — canonical fields; the substrate-native projection is informative
-

05. Comparison with SAFe

Mapping RENAR (the standard for **requirements**) onto SAFe 6.0 (the standard for **scaled agile coordination**). The documents are compatible but have a different scope: SAFe regulates how teams coordinate work at scale; RENAR regulates what a requirement is and how it is verified. This chapter is for teams that already run on SAFe (an enterprise multi-team ART is the typical case) and want to keep their SAFe ceremonies while adding RENAR artifacts as the primary Source of Truth for requirements.

Prerequisites: familiarity with RENAR Core — the conceptual overview written for a human reader and basic SAFe 6.0 terminology (Epic / Capability / Feature / Story, WSJF, PI, ART).

1. Scope: what RENAR regulates, what SAFe regulates

RENAR and SAFe overlap at the level of *work artifacts* (Feature, Story) but solve different problems.

Aspect	SAFe 6.0	RENAR
Standard type	Scaled Agile coordination framework	Requirements-engineering standard
Primary artifact	Epic / Capability / Feature / Story	TZ → ADAPT → BR → SR → SPEC → TC
What it regulates	Cadence, roles, ceremonies, flow	Schema, lifecycle, verifiability, drift control
Artifact substrate	Choice of task-management tool (Jira / Rally / ADO / other)	Substrate-agnostic; normatively — the V1–V6 capabilities (glossary §2.7)
Quality checkpoints	Definition of Done at the Feature level	QG-0 (readiness to start) + QG-2 (verified)
AI-nativeness	Does not regulate how to work with AI	AI as a full participant (adversarial expertise, evaluating scenarios, judge models)

Key principle: SAFe and RENAR are compatible. SAFe says "how to coordinate teams on an ART", RENAR says "what must be true about each requirement for it to count as **verified**". A Feature in SAFe ≡ an SR in RENAR — this is **one and the same artifact**, described from different angles.

2. Master mapping table

SAFe artifact	RENAR artifact	Where it lives	Owner	Acceptance criteria
Strategic Theme	(outside RENAR scope — corporate strategy)	strategic docs	Executive	business level

Portfolio Epic	A group of BR for one system	<code><system>.req/br/</code> aggregated	Lean Portfolio Mgmt	All BR of the group belong to a verified set version
Capability	Subsystem BR (if the subsystem has its own stakeholder)	<code><subsystem>.req/br/</code>	Solution Architect	All subsystem BR belong to a verified set version
Program Epic	Optionally — a large initiative inside the ART, aggregated SR	<code><system>.req/sr/</code> aggregated	Product Manager	The target outcome metric achieved
Feature	SR (or a linked group of SR)	<code><subsystem>.req/sr/</code>	Product Owner	TC from <code>verified-by</code> green with <code>last-run.set-version</code> = the current set version (QG-2)
Story	TR (Task Requirement)	<code><subsystem>.req/tr/</code> or task tracker (Jira, Linear, GitLab Issues)	Team	All AC met, evidence recorded, code merged
Enabler Epic	SPEC-AI / SPEC-ARCH / SPEC-OPS	<code><system>.req/specs/</code>	Architect	Eval metrics within thresholds; baseline recorded
Spike	A research TR + a Decision in the decision log	<code><subsystem>.req/tr/</code> + decision log	Team	Decision recorded and linked to the originating SR
Defect	TR with <code>parent: SR-NN</code> (the violated SR)	task tracker + auto-derived <code>children[]</code> in the SR	Team	Negative TC passes, regression test added

Forbidden mappings: INT-SR is a deprecated term (§4.14.1 Terms), replaced by `SR` with `constrained-by: [SPEC-INT-N]`. See §6 below.

3. WSJF prioritization for RENAR

SAFe uses **WSJF (Weighted Shortest Job First)** as its prioritization framework. RENAR adapts it for the BR / SR level.

3.1 Formula

$$\text{WSJF} = (\text{User-Business Value} + \text{Time Criticality} + \text{Risk Reduction \& Opportunity Enablement}) / \text{Job Size}$$

Each component is rated on a 1-20 scale (modified Fibonacci); WSJF is a relative metric and is only meaningful when comparing BR/SR within a single backlog.

3.2 BR frontmatter extension

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

The `prioritization` field is OPTIONAL in Core RENAR and REQUIRED on an ART that applies SAFe.

3.3 When it applies

- **Mandatory** for BR with `priority: must` in projects coordinated through an ART.
- **Recommended** for all BR in a backlog that goes through PI Planning.
- **Not applied** for SR — an SR inherits the priority of its parent BR. Reshuffling SR within a single BR is the Product Owner's decomposition task, not WSJF.

3.4 Alternatives

If a team does not use SAFe / WSJF, RENAR permits other frameworks:

- **MoSCoW** (Must / Should / Could / Won't) — the simplest one, for small projects. Already present as the `priority` enum in RENAR Core.
- **RICE** (Reach × Impact × Confidence / Effort) — for product-driven teams (typically B2C).

RENAR regulates **the field and its schema**; the choice of framework is up to the project. See also [reference/02-schemas.md](#) for the permitted values of `prioritization.framework`.

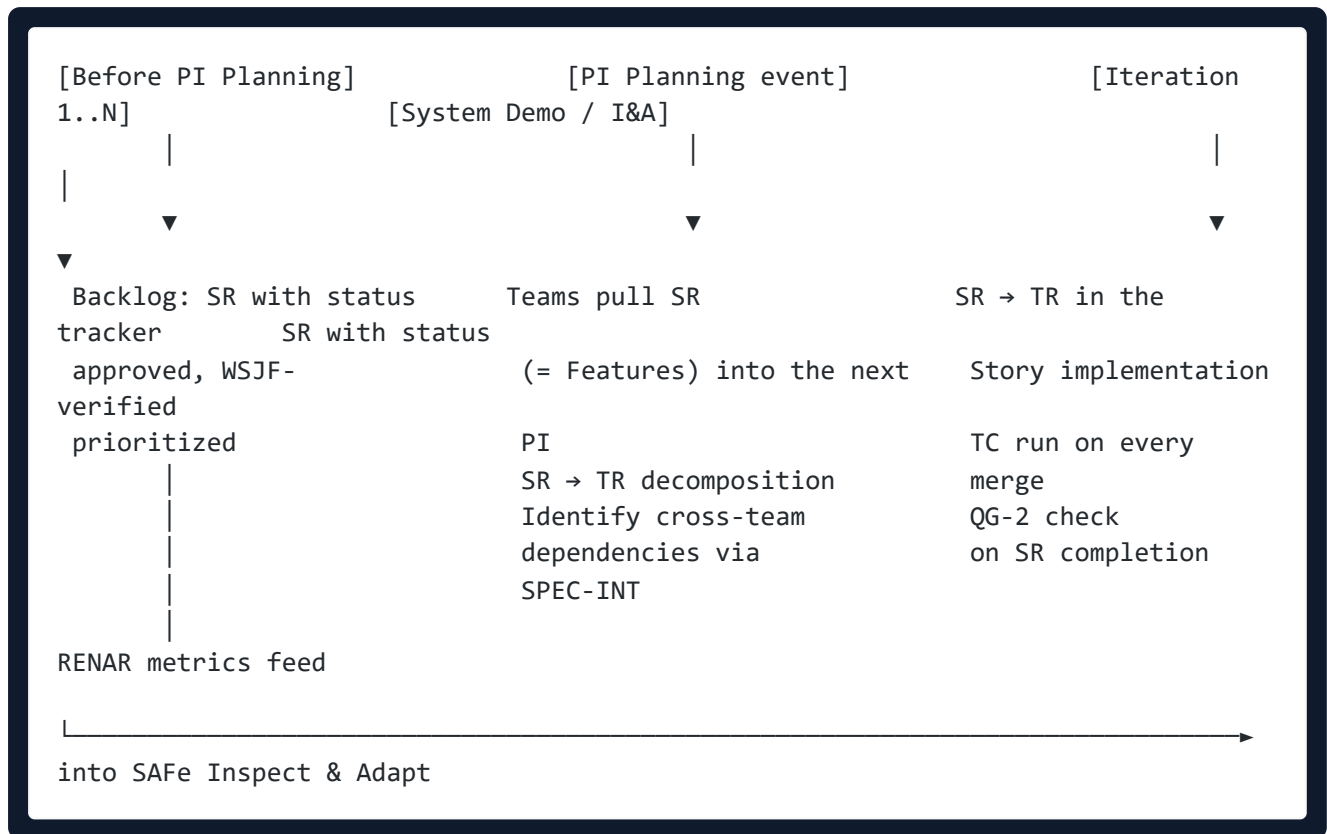
4. PI Planning integration

4.1 What a PI is

A **Program Increment (PI)** is a fixed time box (usually 8-12 weeks) in SAFe during which an ART commits to a set of Features from the shared backlog. PI Planning is a two- or three-day event before each PI,

where teams jointly fix their commitment.

4.2 Where RENAR artifacts enter the PI flow



4.3 RENAR artifacts in each ceremony

SAFe ceremony	RENAR input	RENAR output
Backlog refinement	BR / SR in the draft or in the approved set version	A refined ADAPT, an updated WSJF
PI Planning	WSJF-sorted SR backlog, ADAPT docs	Commitment to SR in the PI; SPEC-INT for cross-team
Iteration Planning	SR + decomposed TR	TR in approved
System Demo	SR of a verified set version	Evidence from the verification entry and the TC last-run
Inspect & Adapt	RDLT, Coverage Velocity, Hallucination Rate metrics	Backlog / process adjustments

5. ART coordination and roles

5.1 SAFe roles ↔ RENAR responsibilities

SAFe role	RENAR responsibility
RTE (Release Train Engineer)	Coordinates cross-team dependencies via SPEC-INT; owner of the PI Objectives ↔ RENAR metrics mapping
Product Manager	Owner of portfolio Epics = groups of BR at the system level
Product Owner	Owner of Features = SR at the team level; accountable for QG-0 (version approval) and QG-2 (version verification)
System Architect	Owner of SPEC-* (especially SPEC-ARCH, SPEC-INT); consulted during BR → SR decomposition
Tech Lead	Accountable for QG-2 at the SR level; owns the team's TR backlog
Business Owner	Approver of BR / ADAPT from the business side
Solution Architect	Owner of BR at the subsystem level (when the subsystem has its own stakeholder)
Scrum Master	Owner of ceremonies; does not own RENAR artifacts directly

5.2 Cross-team coordination

In a typical SAFe organization with several teams on a single ART:

- **Each team** owns its SR / TR in `<subsystem>.req/`.
- **The RTE / Solution Architect** own the SPEC-INT in `<system>.req/specs/int/` — the shared integration contracts between subsystems.
- **Changing a SPEC-INT** requires cross-team agreement (QG-0 from every affected team).

Rule: ART-level roles (the RTE) **do not edit** SR in subsystems directly. Coordination flows through SPEC-INT, which is explicitly **constrained-by** for every affected SR.

6. Cross-team dependencies via SPEC-INT

When a Feature in one subsystem blocks / depends on another:

6.1 An SR with a dependency

```
# In <subsystem-a>.req/sr/SR-05.md
---
id: SR-05
parent: BR-12
title: "User registration via corporate email"
constrained-by:
  - id: SPEC-INT-01
    ref: "specs/int/SPEC-INT-01-auth-handshake.md"
verified-by:
  - TC-23
```

6.2 SPEC-INT as a contract

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

6.3 Breaking changes in SPEC-INT

Any change to `SPEC-INT.version` with breaking semantics (§4.11 [Drift classes](#)) requires:

1. An ADAPT-level discussion (why we are changing it).
2. Agreement from every `participant` (QG-0 from each team).
3. A migration plan for existing implementors (how the old SR will stay valid or be adapted).

The RTE **MUST** check SPEC-INT consistency across subsystems regularly (usually once per sprint) — this is part of Inspect & Adapt and feeds into the conformance self-assessment (§13 [Conformance](#)).

7. Definition of Done at each level of the hierarchy

DoD is the set of **formal** conditions that are checked automatically (substrate hooks). Each level of the hierarchy has its own DoD.

Level	RENAR artifact	DoD criterion
Strategic Theme	—	(outside RENAR scope)
Portfolio Epic	A group of BR	All BR of the group — in a verified set version, KPI impact confirmed via an outcome metric
Capability	Subsystem BR	All BR — in a verified version; outcome metric measured
Feature	SR	QG-2 passed: every TC from <code>verified-by</code> has <code>last-run.result = pass</code> with <code>last-run.set-version</code> = the set version

Story	TR	All AC met, evidence recorded, code merged, automated test exists
Enabler	SPEC-AI / SPEC-ARCH / SPEC-OPS	Eval runs cleared the thresholds (for SPEC-AI); baseline recorded (for all)

Key point: at the Feature level the DoD in SAFe **coincides** with QG-2 in RENAR. There are not two different "definitions of done" — it is one and the same gate, described from different angles.

8. Built-in Quality ↔ RENAR mechanisms

The SAFe Built-in Quality principle: quality is built into the process, not ad-hoc. RENAR implements Built-in Quality through normative mechanisms:

SAFe Built-in Quality practice	RENAR mechanism
Continuous Integration	Substrate hooks + CI on every change in requirements (§13 Conformance); reconciliation hook (drift detection, §4.11)
Test-First	TC norms are created before implementation and approved in the same set version as the SR; QG-0 of the set requires a pair of TC norms for every statement
Refactoring	The continuous reconciliation hook (§2.4.2); detects drift between the requirements and the code
Pairing / Mobbing	AI generator + AI critic (§5.2 Roles) — pair generation/review as a normative role
Definition of Done	QG-2 as a formal gate, checked automatically (§10 Lifecycle and QG)
Version Control	The V1 capability (immutable history) without tying to a class of storage environment
Automation	The V2-V6 capabilities — all verifications are automatable

RENAR **does not prescribe** the instrument (Jenkins / GitLab CI / GitHub Actions / Tekton) — only **what** must be checked (the capability), not **how**.

9. RACI of an artifact lifecycle (with SAFe roles)

The full lifecycle of a RENAR artifact with responsibility distributed across SAFe roles.

Activity	Responsible	Accountable	Consulted	Informed
TZ import	AI agent	System Architect	Business Owner	Team, RTE
TZ → ADAPT decomposition	AI generator	System Architect	Stakeholder, AI critic	RTE, Team
Decomposition → BR	AI generator	Product Owner	Business Owner, AI critic	RTE, Team
WSJF prioritization	Product Manager	Product Owner	RTE, Stakeholder	Team

BR → SR decomposition	AI generator	System Architect	AI critic	Team, RTE
SR → SPEC-* decomposition	System Architect	System Architect	AI critic, Tech Lead	Team
TC generation	AI agent	Test Architect	—	Team
QG-0 approval	System Architect	Tech Lead	AI critic	Business Owner, RTE
SR selection for the PI	Product Owner	RTE	Team (capacity)	Stakeholder
SR → TR decomposition	Team	Product Owner	Tech Lead	RTE
TR implementation	Developer	Tech Lead	—	Team
TC run (automated)	Substrate hook	—	—	Team
QG-2 (version verified)	System Architect	Tech Lead	—	Business Owner, RTE
System Demo	Team + RTE	Product Owner	Stakeholder	RTE, Executive
delta-TZ approval	System Architect + Stakeholder	Product Owner	AI impact analysis, RTE	Team
Spot-check of 5 TC (audit)	System Architect	—	—	RTE
Reconciliation MR	AI reconciler agent	System Architect	—	Team

10. PI Objectives ↔ RENAR metrics

PI Objectives are SMART outcomes for the quarter. RENAR metrics ([§12 Metrics, reference/02-schemas.md](#)) feed into PI Objectives:

PI Objective (example)	RENAR metric
"Reduce time from TZ signing to first commit to < 2 days"	RDLT (Requirement Decomposition Lead Time)
"Reach Coverage Velocity ≥ 60% per sprint"	Coverage Velocity (% of SR with status: verified per sprint)
"Lower the Hallucination Rate in new requirements to < 2%"	Hallucination Rate (% of AI-generated statements rejected at review)
"0 disputed requirements at acceptance in this PI"	Dispute Rate at Acceptance
"Drift detection — all SR consistent with code within 24 hours of merge"	Reconciliation Findings per Week (§12.3.10)

This turns RENAR from a "standard for documents" into a **measurable contribution** to ART success. The metrics feed automatically into Inspect & Adapt; the RTE uses them in the retrospective at the close of the PI.

11. What to keep from SAFe, what to replace

For teams migrating to RENAR from an already-running SAFe process:

11.1 Keep as is

- **Cadence** (PI, Iterations) — RENAR does not regulate time; use your own rhythm.
- **Ceremonies** (PI Planning, System Demo, I&A, Daily Standup) — RENAR artifacts fit transparently into these events.
- **WSJF** — keep it for prioritizing BR / SR; it fits into `prioritization.framework: WSJF`.
- **ART, RTE, Scrum Master** — the structure and roles are preserved.
- **PI Objectives** — keep them; the mapping onto RENAR metrics (§10) makes them measurable.

11.2 Replace with the RENAR equivalent

- **Feature description in Jira / Rally / ADO** → an SR with full frontmatter in `<subsystem>.req/sr/`. The tracker record becomes a **mirror** of the RENAR artifact, not the primary source.
- **Acceptance criteria in a Feature** → `verified-by: [TC-NN]` with automatically checked TC.
- **Definition of Done on a Feature** → QG-2 (no two different DoDs — it is one gate).
- **Integration agreements between teams** → SPEC-INT (replaces the INT-SR from older SAFe implementations).
- **Architecture Decision Records (ADR)** → SPEC-ARCH / SPEC-AI / SPEC-OPS (one of the 12 SPEC types, §4.4).

11.3 Add (new in RENAR)

- **ADAPT** — the bridge artifact between the TZ and the requirements hierarchy. SAFe has no direct analog; it is created **reactively** ([standard/07 §7.4.1](#)) — only when the adversarial reviewer returns a "findings present" verdict. On a "no findings" verdict no ADAPT is created, and BR / SR / SPEC are derived from the TZ directly via `source.tz-section` + `source.adversarial-review-ref`. What is mandatory is the adversarial review itself, not the ADAPT.
 - **The AI critic role** — adversarial review of AI generation. Not regulated in SAFe; in RENAR Core it is mandatory for all AI-generated artifacts.
 - **Reconciliation (drift detection)** — continuous reconciliation of requirements against the implementation (relies on V5 — end-to-end version pinning). In SAFe it is a manual reconciliation; in RENAR it is a normative mechanism.
-

12. Negative: what this chapter does not claim

- **RENAR is not a replacement for SAFe.** RENAR regulates *requirements*, SAFe regulates *work coordination*. A team can run on RENAR without SAFe (a small project, a single team) or with SAFe (an enterprise multi-team ART scope).
 - **RENAR is not a planning standard.** Cadence, capacity planning, velocity tracking — all outside the scope. RENAR says "what must be true about a requirement", not "when the team must finish it".
 - **RENAR does not forbid other frameworks.** WSJF, MoSCoW, RICE — all compatible. RENAR fixes `the field prioritization.framework`, not the choice of framework.
 - **RENAR does not regulate the Jira / Rally / ADO workflow.** A tracker-level implementation is a substrate detail; the normative source is `<subsystem>.req/`.
 - **RENAR does not prescribe the PI as mandatory.** A team can run on continuous flow without a PI; in that case sections 4 and 10 of this chapter become informative.
-

13. Resolved decisions for v1.0

- **priority: must does NOT require a WSJF score, even in SAFe projects.** Per §12 of this chapter: RENAR fixes `prioritization.framework`, it does not prescribe the choice of framework. `priority: must` is a RENAR MoSCoW marker ([reference/02-schemas](#)), independent of WSJF. WSJF is an optimization for SAFe teams, not a normative RENAR requirement.
- **Feature/Capability/Story mapping is substrate-specific.** RENAR regulates the closed list of requirement types (BR/SR/TR + 11 SPEC) and does not introduce SAFe Feature levels. Projects that apply SAFe fix the mapping in a substrate-native `safe-mapping/` manifest (see [reference/02-schemas](#) — informative extension).
- **PI Objectives are informative, outside RENAR scope.** The PI is a SAFe coordination artifact; RENAR does not regulate it. If a team wants traceability, an informative `cross-link.pi-objective-id` field in the BR frontmatter is recommended — it is not conformance-gating, but it eases RTE-side queries.
- **Inspect & Adapt: metrics are automated substrate-natively.** Per §10.13 Logging + §12 — the substrate MUST surface COVERAGE / audit-trail substrate-natively. The RTE uses the same data through the query API; manual extraction is an anti-pattern (drift).

13.1 Deferred to v1.1 (phase 8 backlog)

- **An AI heuristic for estimating the WSJF Job Size field** (a 1–20 scale by the number of AC, TC complexity, and code surface area). Not regulated in v1.0; specific to the SAFe–RENAR binding. The extended informative mapping — [reference/11](#).
-

14. Relationship to other chapters

- [00-quickstart](#) — the basic RENAR cycle without the SAFe overlay.
 - [01-walkthrough](#) — a full example on a small-scale project (without PI Planning).
 - [reference/01-glossary](#) — the exact semantics of BR / SR / SPEC / TR / TC.
 - [reference/02-schemas](#) — frontmatter schemas, including `prioritization`.
 - [standard/04-terms](#) — normative definitions; the mapping onto SAFe is fixed in §4.13.
 - [standard/08-specifications](#) — the closed list of 12 SPEC types, including SPEC-INT.
-

06. Compliance

*RENAR is a requirements-engineering standard; it is not a compliance standard in its own right. But RENAR provides the **traceability infrastructure** that makes conformance to other standards (ISO 27001, GDPR, FZ-152, EU AI Act, and others) automatically verifiable. This chapter is a mapping of RENAR artifacts onto 8 key compliance frameworks, plus self-assessment checklists, plus a list of auto-generated artifacts for the auditor.*

Prerequisites: *RENAR Core, [reference/02-schemas.md](#), [reference/03-ai-risk-register.md](#).*

1. Compliance principles in RENAR

1.1 Compliance frontmatter. Every requirement with a regulatory rationale **MUST** carry a `compliance` field in its `frontmatter` — you cannot trace conformance through an external table that is not linked to the artifact. The field is repeatable (one requirement **MAY** close controls across several standards):

```
compliance:
- { standard: "ISO 27001:2022", control: "A.5.34", rationale: "Privacy and
  protection of PII" }
- { standard: "GDPR", article: "Art.32", rationale: "Security of processing" }
- { standard: "FZ-152", article: "Art.19", rationale: "Measures to ensure the
  security of personal data" }
```

1.2 Data classification. Every BR/SR that operates on data **MUST** declare a classification. When `contains-pii: true`, SR-encryption + SR-audit-log + SR-erasure automatically become mandatory.

```
data-classification:
contains-pii: true           # GDPR / FZ-152 trigger
contains-financial: false   # PCI-DSS trigger
contains-health: false      # HIPAA / FZ-152 special categories
contains-children-data: false # COPPA trigger
retention-days: 1095
data-residency: ["RU", "EU"]
```

1.3 Traceability as the audit foundation. The chain `BR → SR → SPEC → TC → implementation → CI run` is itself the audit trail. The auditor opens the coverage report and sees: which requirements close a control; which TCs verify them; `last-run.date` ; `last-run.set-version` (the set version the run passed on). Audit time: 1–2 days instead of 2–3 weeks. The reconciliation hook (drift detection) guarantees that the evidence has not gone stale between audits.

2. ISO/IEC 27001:2022 — Information Security

ISO 27001:2022 Annex A control	RENAR artifact
A.5.7 Threat intelligence	SPEC-SEC with a threat model; SR with <code>compliance: A.5.7</code>
A.5.8 Information security in project management	RENAR itself as process compliance
A.5.34 Privacy and protection of PII	SR with encryption + <code>data-classification.contains-pii: true</code>
A.6.3 Information security awareness	The onboarding process includes reading RENAR
A.8.1 User endpoint devices	SR with device requirements (if in scope)
A.8.5 Secure authentication	SR-AUTH-* + SPEC-SEC with an authentication flow
A.8.7 Protection against malware	SR + adversarial review (the AI critic)
A.8.10 Information deletion	SR with deletion logic + <code>retention-days</code>
A.8.16 Monitoring activities	SR with logging + SPEC-OPS audit-log
A.8.24 Use of cryptography	SR with an explicit crypto algorithm + ISO 25010 Security
A.8.25 Secure development life cycle	SENAR + RENAR as evidence
A.8.28 Secure coding	TC with security checks; SPEC-SEC with STRIDE coverage
A.12.1.2 Change management (legacy 27001:2013)	Delta-TZ + Impact Analysis (§7.6)

Audit deliverable. Auto-generates a conformance report: scope (BR/SR/TC counts with `compliance.iso27001`) + Coverage by Annex A (control ↔ mapped SR ↔ status). The normative mechanism is capability V4 reporting from versioned artifacts.

3. GDPR (Regulation EU 2016/679)

GDPR Article	RENAR artifact
Art.5 Principles	BR explicitly states the lawful basis
Art.6 Lawfulness	BR with <code>gdpr.lawful-basis: consent / contract / legal-obligation / vital-interests / public-task / legitimate-interests</code>
Art.7 Conditions for consent	SR with a consent-management workflow
Art.13 Information to data subject	SPEC-UI with a privacy-notice screen
Art.15 Right of access	SR with export-user-data
Art.16 Right to rectification	SR with edit-user-profile

Art.17 Right to erasure	SR with delete-user-data + propagation to linked entities
Art.18 Right to restriction	SR with suspend-processing
Art.20 Right to data portability	SR with export in a machine-readable format
Art.25 Data protection by design and by default	data-classification is mandatory at the BR level
Art.30 Records of processing activities	Auto-generated from BRs with contains-pii
Art.32 Security of processing	SR with encryption + access control
Art.33 Notification of personal data breach	SR with a breach-notification workflow + a 72h timer
Art.35 DPIA	For high-risk — a separate dpia/<feature>.md

GDPR frontmatter:

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

DPIA. For high-risk processing — <system>.req/dpia/DPIA-NN-<slug>.md with frontmatter (id , type: DPIA , gdpr-article: 35 , related-br[] , risks-identified , mitigations) and sections: the processing operation, necessity, risks, mitigation measures, and sign-off with the DPO.

4. FZ-152 "On Personal Data" (RF)

FZ-152 Article	RENAR artifact
Art.5 Processing principles	BR with a stated purpose (business-context.business-goal)
Art.6 Processing conditions	BR with a lawful-basis
Art.9 Consent to processing	SR with consent management
Art.13.1 Storage within the RF	data-classification.data-residency: ["RU"] for Russian clients

Art.14 Right of access to information	SR with export-user-data
Art.15 Right to rectification	SR with edit-user-profile
Art.16 Deletion	SR with delete-user-data
Art.18 Operator obligations	The audit trail via RENAR traceability
Art.18.1 Localization of RF citizens' personal data	data-residency is mandatory
Art.19 Protection measures	SR with encryption + access control + ISO 25010 Security
Art.21 Breach notification	SR with a breach-notification workflow
Art.22 Notification to Roskomnadzor	operational, outside RENAR scope

Data residency enforcement. For projects with Russian clients, `data-residency` is mandatory. The substrate hook checks: when `data-residency: ["RU"]`, all upstream SRs (storage, backups, replication) MUST have evidence of storage within the RF; adding a foreign jurisdiction for an RU-resident BR → a block at QG-0.

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

The EU AI Act classifies AI systems by risk level. RENAR requires that the class be stated explicitly:

```
ai-act:
  risk-class: limited                # prohibited | high | limited | minimal
  rationale: "Generative AI for document drafting; no autonomous decisions
affecting users' legal rights"
  high-risk-domain: false           # true → a conformity assessment is
required
  general-purpose-ai: true          # GPAI – Claude / GPT, etc.
```

High-risk AI requirements (Art.9–15) — for the `high` class (financial scoring, hiring, public services, medical diagnostics):

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

GPAL obligations (Art.51–55). If a General-Purpose AI Model (Claude, GPT, Gemini) is used: `ai-provenance` in the frontmatter of any AI-generated artifact (model id, version, prompt hash); the model's technical document (if it is a GPAL with systemic risk) — an external document + a reference from SPEC-AI.

6. NIST AI RMF 1.0 (US jurisdiction)

NIST AI RMF Function	RENAR mechanism
Govern	RENAR + SENAR roles + compliance frontmatter
Map	BR with <code>business-context</code> , <code>data-classification</code> , <code>ai-act.risk-class</code>
Measure	Eval-tests with metric thresholds + RENAR metrics (Hallucination Rate, Coverage Velocity)
Manage	Lifecycle with QGs + reconciliation hook + AI risk register

RMF-specific extensions (optional for US projects; not in conflict with ISO/IEC 23894 §7):

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

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

Guidance on AI risk management. It does not certify, but it provides a structured framework for managing the risks of AI systems. Compatible with NIST AI RMF and the EU AI Act.

ISO/IEC 23894 clause	RENAR artifact
§6.4.1 Risk identification	AI risk register (reference/03)
§6.4.2 Risk analysis	SPEC-AI with a risk-assessment section
§6.4.3 Risk evaluation	A threshold for each risk in <code>eval-tests</code>
§6.4.4 Risk treatment	SR-mitigations + post-mitigation evaluation
§6.4.5 Communication and consultation	RACI matrix (05-safe-comparison §9)
§6.4.6 Monitoring and review	Reconciliation hook (drift detection)

Conformance criteria: every AI-generated artifact carries `ai-provenance` ; for each AI use case, SPEC-AI documents the identified risks (hallucination, bias, robustness), the mitigations (eval thresholds, adversarial review, human-in-the-loop), and the residual risks; the risk register is updated whenever SPEC-AI changes (reconciliation catches the drift).

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

An extension to ISO/IEC/IEEE 12207. A convenient framework for AI Act Art.11 technical documentation.

ISO/IEC 5338 process	RENAR stage
Stakeholder needs and requirements definition	TZ + ACTZ (clarification protocols) + ADAPT + BR
Architecture definition	SPEC-ARCH + SPEC-AI
Design definition	SPEC-AI (model card, prompt design, RAG)
Implementation	TR + implementation
Verification	TC (<code>tc-type: eval</code> for AI)
Validation	AT — acceptance tests from the effective TZ (§9.19) + spot-check
Operation	Reconciliation hook (drift detection, §4.11)
Maintenance	Delta-TZ + ACTZ + ADAPT iteration
Disposal	Retirement workflow (outside Core RENAR scope)

Conformance evidence: SPEC-AI for each AI use case with a full model card; eval-tests bound to SPEC-AI via `verifies[]` ; `ai-provenance` recorded; drift detection active (V6).

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

If the project handles payment-card data (CHD):

PCI-DSS v4 requirement	RENAR artifact
Req.1 Network security controls	SPEC-OPS with network segmentation
Req.3 Protect stored CHD	<code>contains-financial: true</code> + SR with encryption-at-rest
Req.4 Encrypt CHD transmission	SR with TLS + SPEC-API requirements
Req.6 Develop secure systems	RENAR + SENAR as process compliance
Req.7 Restrict access by need to know	SR with RBAC + SPEC-SEC access-control matrix
Req.8 Authenticate access	SR-AUTH-* + MFA where mandatory
Req.10 Log and monitor access	SR with an audit trail + SPEC-OPS observability

Req.11 Test security regularly	TC <code>tc-type: security + pen-test</code> (outside RENAR scope)
Req.12 Information security policy	RENAR + SENAR as a documented policy

CHD scope minimization. The substrate hook checks that new SRs with `contains-financial: true` explicitly justify the need to store CHD — without that justification, the requirement is halted at QG-0.

10. Self-assessment checklists

Short yes/no checklists for compliance teams. The full printable kit for a RENAR manifest — [reference/08](#).

ISO 27001:2022 — every BR with `contains-pii: true` has an SR closing A.5.34; the SoA is documented; every in-scope control has ≥ 1 verifying SR; the coverage report is green; an auditor goes from a control to a passing TC in under 5 minutes.

GDPR — all PII is in the Records of Processing (auto-generated); the lawful basis is stated per processing activity; data-subject rights Art.15–22 are verifiable through SR + TC; a DPIA is performed for high-risk; cross-border transfers are justified (SCCs/adequacy).

FZ-152 — requirements with PII of RF citizens carry `data-residency: ["RU"]`; the substrate hook checks the upstream storage SRs; consent is implemented in an SR with TC evidence; the Art.19 protection measures are covered by an SR with a passing TC.

EU AI Act — every SPEC-AI carries `ai-act.risk-class`; for `high`: Art.9–15 evidence is in the substrate; human oversight at QG-0/QG-2 + spot-check; technical documentation is auto-generated; `ai-provenance` for GPAI components.

NIST AI RMF — all 4 functions (Govern/Map/Measure/Manage) have evidence; an AI Governance Lead is defined in roles; the eval baseline is recorded; an incident-response plan is linked to SPEC-AI.

ISO/IEC 23894 — the AI risk register is filled out and linked to SPEC-AI; every risk has a mitigation in an SR; residual risks are accepted by the owner; V6 is active.

ISO/IEC 5338 — SPEC-AI for each AI use case with a model card; eval-tests bound via `verifies[]`; `ai-provenance` recorded; V6 is active.

PCI-DSS — SRs with `contains-financial: true` have encryption (at-rest + in-transit); the CHD scope is minimized; RBAC is in SPEC-SEC; the audit-trail SR covers all CHD-touching flows; security TCs run regularly.

10.9 RENAR-conformance self-assessment (in an hour)

A project declares its **own** RENAR-conformance level via the `RENAR-CONFORMANCE.yaml` manifest ([standard/13 §13.4](#)). Quick checklist (yes/no, single pass):

- The requirements substrate provides all of V1–V6 ([§11.4.1](#) — Notion/Google Docs do not qualify).
- Each TZ has an issued AR (adversarial-review record); every ADAPT produced by a "findings present" verdict is in status approved with the **Architect's signature** ([§7.5](#)). There may be no ADAPT at all (a "no findings" verdict) or several (findings at different stages) — both are regular cases.
- A mandatory client signature on the ADAPT is **not** declared: obligations to the client are carried by the ACTZ. Requiring a client signature on the ADAPT is permitted only as `declared-`

`stricter` , never as base conformance (§13.3.3).

- Every finding in status `resolved` carries a `decided-in` pointing at a clause of a **signed** ACTZ; no signed ACTZ is left unreflected in an ADAPT (§7.4.5, §7.13).
- Acceptance runs against the **effective TZ** (the initial TZ + every signed ACTZ, §7.14), not against the ADAPT.
- The ATs were derived by an isolated agent: a model different from the primary agent's, with no access to the ADAPT / BR / SR / SPEC / TC / code; each AT's `tz-version` matches the current revision of the effective TZ; every AT is `passing` before submission (§9.19, §10.4.3).
- No TC counts as evidence without a red history; the exception is a TC of class `implementation-originated` , where a killed mutant serves as the compensation (§9.18.2).
- Requirements of class `implementation-originated` describe internal technical detail only: client-observable behavior is not legalized through this class (§6.13.2).
- All 12 SPEC types are supported natively (a declaration of "type not used" is permitted, "not supported" is not).
- Every normative statement covered by a TC has a paired negative TC.
- QG-0 / QG-1 / QG-2 are declared `required` .
- The manifest contains `renar-version` + `senar-version` + `level` + confirmation of the mandatory clauses (reference/08 §2).
- `assessment-date` is recent, `next-assessment-due` is not overdue.

All 13 are `yes` → the project is conformant to the declared `level` . At least one `no` → the manifest is not issued (§13.5.2).

Four items of the list — 3, 6, 7 and 8 — carry one point: **an obligation to the client cannot arise out of an engineering document the client never read, nor out of the code, nor out of a test written by the same agent and the same model as the implementation.** Their sources differ, and are worth naming: the client signature on an ADAPT is not required — §13.3.3 and §7.5; AT generator isolation — §9.19.2 with the control point at §10.11.1; red history — §9.18.2; the bounds of the `implementation-originated` class — §6.13.2. Of the four, only the first belongs to the closed register of mandatory clauses in §13.3; the other three are declared mandatory in their own chapters.

A filled-in example (RENAR-2, self-assessment) — full schema in §13.4.2:

```
renar-version: "1.1"
senar-version: "1.0"
manifest-version: 1
manifest-id: "CFM-2026-014"
level: "RENAR-2"
level-target: "RENAR-3"
assessment-mode: "self"
assessment-date: "2026-05-20"
assessor: { id: "architect-team-lead", role: "architect", signature-ref: "<pointer>" }
next-assessment-due: "2026-08-20"
mandatory-clauses-confirmed:
  sot-inversion: true
  substrate-v1-v6: { v1: true, v2: true, v3: true, v4: true, v5: true, v6: true }
  adapt-per-tz: true
```

```

spec-types-closed-list: true
tc-pos-neg-pairing: true
quality-gates-closed-list: true
closed-lists-backward-findings: true
quality-gates: { qg-0: required, qg-1: required, qg-2: required, qg-3: absent,
qg-4: absent }
spec-types-supported: ["SPEC-ARCH", "SPEC-API", "SPEC-DATA", "SPEC-INT", "SPEC-
PROC", "SPEC-UI", "SPEC-AI", "SPEC-SEC", "SPEC-OPS", "SPEC-TEST", "SPEC-DOC", "SPEC-UC"]
exceptions: []
replaced-by: null

```

11. Auto-generated compliance artifacts

Artifacts generated from existing requirements for auditors:

Artifact	Source	Contents
Records of Processing (GDPR Art.30)	BR/SR with <code>contains-pii: true</code>	Data categories, lawful basis, retention, recipients
Statement of Applicability (ISO 27001)	BR/SR with <code>compliance: ISO 27001:2022</code>	Annex A controls in/out of scope, justification
Data Inventory	All <code>data-classification</code> records	Categories, residency, retention, owners
AI System Cards	SPEC-AI	Model card in NIST AI RMF / AI Act format
Audit-trail report	Traceability BR → SR → SPEC → TC → last-run	The chain from control to evidence
DPIA Index	The <code>dpia/</code> folder	A list of DPIAs + their DPO sign-off status
AI Risk Register Snapshot	AI risk register	All identified risks + mitigations + residual

Substrate-native reporting: aggregation of evidence from versioned artifacts (V4) into a compliance report on the assessor's request.

Evidence pack — templates (informative; full E2E — [guide/09 §E3](#)). GDPR Art.15 trace bundle — a `Control | BR/SR | SPEC | TC | last-run` table + lawful basis + retention + a link to the manifest. FZ-152 Art.14 — a mapping table `Requirement ↔ RENAR artifact ↔ Evidence field`. ISO 29148 trace excerpt — [reference/07](#) (fill in "RENAR frontmatter" for each in-scope SR). Pre-audit self-assessment — [reference/08 §2](#).

12. What RENAR does NOT cover in compliance

RENAR is infrastructure for compliance, but not a **substitute** for: the DPO (Data Protection Officer — a human role with legal accountability); legal review of contracts and privacy notices; pen-testing the

implementation; bug bounty / vulnerability management; physical security (data centers, the office); operational security (key rotation, incident response, the monitoring runbook); cyber insurance; regulatory filings (Roskomnadzor notifications, GDPR registration with a DPA, AI Act conformity assessment).

RENAR helps you do compliance **during requirements engineering**. Operational compliance consists of separate processes that *reference* RENAR artifacts as evidence.

13. Resolved decisions for v1.0

- **Compliance frontmatter — conditionally mandatory.** By default it is optional; it is mandatory when `contains-pii: true` (GDPR/FZ-152 trigger) or `contains-phi: true` (HIPAA trigger). An artifact with PII and no compliance frontmatter is non-conformant (§13.3 extensions via a manifest `declared-stricter`).
- **DPIA — mixed model.** The formal DPIA is an external document (legal owns it); the RENAR artifact `dpia/<slug>.md` holds a machine-readable summary + a pointer to the formal document. Separation of concerns while preserving traceability.
- **Data residency — outside RENAR scope.** Per §1.3 (3), tech stack / infra are out of scope. Enforcement is via DevOps-level controls (network policies, cloud regions). RENAR records the **requirement** (`data-classification.data-residency: ["EU"]`), not the enforcement.
- **Custom industry frameworks** (CB RF fintech, HIPAA healthcare, etc.) — separate documents (industry-specific addenda as `guide/06-compliance-<industry>.md` or external documents; they **MAY** be `declared-stricter` on top of RENAR conformance).

Deferred to v1.1 (phase-8 backlog): auto-export of evidence for Schrems II and EU ↔ RF transfers — partially addressed by the `cross-border-transfer` flags and links to SCCs, but full automation is unattainable (which SCCs apply is a lawyers' call). Owners: the legal-tech / adapters pairing.

14. Relationship to other chapters

- [00-quickstart](#) — the basic RENAR cycle without the compliance layer.
 - [05-safe-comparison](#) — RACI with SAFe roles (including the DPO as Consulted).
 - [reference/02-schemas](#) — frontmatter schemas: `compliance` , `data-classification` , `gdpr` , `ai-act` .
 - [reference/03-ai-risk-register](#) — the AI risk register structure.
 - [reference/04-ai-style-guide](#) — the AI-provenance style.
 - [standard/04-terms](#) — SPEC-SEC, SPEC-AI, SPEC-OPS terminology.
 - [standard/13-conformance](#) — RENAR conformance levels (≠ compliance: RENAR-N assesses the maturity of the *process*, compliance is conformance to *external norms*).
-

07. Failure modes

A systematic survey of every known way a RENAR project can break down: technical drift between artifacts and implementation, AI-specific risks, and (most importantly) organizational patterns where the process exists on paper but does not work. Drift classes are normalized in standard/00 §0.3; AI risks — in reference/03. For each failure mode — symptom, how to detect, how to prevent, how to recover.

Prerequisites: RENAR Core, [reference/03-ai-risk-register.md](#).

1. Map of failure modes

Three classes of problem:

Class	Where it lives	How it surfaces
Drift	A mismatch between different representations of the same entity (frontmatter ↔ DB, requirement ↔ code, TC ↔ requirement)	Reconciliation hook (drift detection, §4.11)
AI risks	Properties of AI generation (hallucination, bias, injection, model drift)	Adversarial review + eval tests + AI risk register (reference/03)
Organizational	A mismatch between the formal process and the team's real practices	Behavioral signals: the approval pattern, the frequency of disputes, the frequency of bypass

Drift and AI risks are caught by substrate mechanisms. Organizational failures are caught by no substrate; they require human-level process reviews. This chapter covers all three.

2. The 8 drift classes

For each: symptom (how it looks from the outside), detection (how to catch it automatically), prevention (how to avoid it), recovery (what to do once it has happened).

2.1 Schema drift

Symptom: The fields in an artifact's frontmatter diverge from what the substrate expects / supports.

Detection: On every change to an artifact, the substrate validates the frontmatter against the schema ([reference/02-schemas.md](#)). On a divergence, integration is blocked (QG-0 fails).

Prevention: The schema is the single source of truth (closed list); it is not edited within the project. Schema changes happen only through a change to the full RENAR Standard.

Recovery: Roll the frontmatter back to a schema-valid state; if the change is genuinely needed, open an RFC for a standard change.

2.2 Lifecycle drift

Symptom: States (draft / set version `N.M` / TR and ADAPT statuses) and quality-gate names are understood differently across subsystems or across teams.

Detection: Compare the status transitions in the audit trail against the normative state machine ([standard/10-lifecycle-qg](#)). Anomalies (a transition without the corresponding pre-conditions) are flagged.

Prevention: Transitions are performed by the substrate mechanism, not by manual frontmatter edits. Capability V3 (state-machine enforcement).

Recovery: Roll back the illegitimate transition; re-run QG-0 / QG-2 through the correct mechanism.

2.3 Source-of-truth drift

Symptom: The same entity is edited in two places (for example, both in the `.req` directory and in the Jira tracker). The versions diverge.

Detection: Periodic reconciliation between the substrate and the tracker; the diff reveals the divergences.

Prevention: At any moment in time, **exactly one SSoT substrate is chosen** for the project. The tracker is a derived view, not the Source of Truth. The substrate hook blocks tracker-only changes to requirements.

Recovery: Declare one substrate the winner; merge the second into the first; stop editing in the second until migration.

2.4 Implementation drift

Symptom: Code in the implementation references an SR that no longer exists (deprecated, removed, renamed). Or: the SR exists, but the implementation has drifted away from it (the behavior does not conform).

Detection: Reconciliation hook (drift detection):

- Forward: walk from a requirement → find the implementing code → run the TC.
- Backward: walk from the code → find references to SR / TC → check that they belong to a verified set version.

Prevention: Requirement IDs are **immutable** — renaming is forbidden. Deprecated requirements stay in the repository with status `deprecated` ; they are not deleted.

Recovery: Open a delta-TZ that explicitly adopts the current implementation (or, conversely, requires the code be rolled back into conformance with the requirement).

2.5 Terminological drift

Symptom: "Verified", "implemented", "approved" mean different things to different people / teams.

Detection: Code-review checklist: "a term not from the glossary was used?" — a flag. Likewise, the substrate validator checks that the values of enum frontmatter fields come only from the closed list.

Prevention: The glossary is the single source of terms ([reference/01-glossary](#)). Each term = exactly one lifecycle state.

Recovery: Audit all project artifacts for the use of out-of-glossary terms; replace them or file an RFC to extend the glossary.

2.6 Order / provenance drift

Symptom: Delta-TZ #2 references an SR that was created in Delta-TZ #1, but application happened in reverse order — the SR did not exist at the moment #2 was applied.

Detection: Delta-TZs are numbered and applied strictly in number order. The substrate hook checks that the upstream delta has already been applied.

Prevention: Delta-TZs cannot be renumbered. Each artifact stores `created-by-order` (the delta-TZ of creation) and `last-modified-by-order` (the last update).

Recovery: Roll back the out-of-order application; re-apply in the correct order.

2.7 TC ↔ requirement provenance drift

Symptom: A TC verifies a requirement, but the norm has already been changed in a new set version — the implementation's `automation.set-version` is lower than the version in which the norm was changed. The test is green, but it checks outdated behavior.

Detection: The coverage report shows a "Stale" category — implementations with an outdated `automation.set-version` ([standard/10 §10.9.4](#)). Reconciliation catches this automatically (via the V5 pin of the set version).

Prevention: A TC has the mandatory field `automation.set-version` — the set version the implementation was written against. QG-2 does not record the verification of a version if at least one implementation is stale or its `last-run.set-version` differs from the version.

Recovery: Re-run the stale TC against the current requirement version; update it if the TC itself is outdated.

2.8 Test-fitting drift

Symptom: An AI agent has a trivial path to turning a failing test green — weaken the pass/fail criterion instead of fixing the code. Without protection, tests drift from "strict checker" to "green void".

Detection: A change to a TC's pass/fail criteria without an explicit `[test-spec-change]` tag is flagged by the substrate. A periodic spot-check of 5 random passing TCs once per sprint.

Prevention:

- An MR / change that modifies pass/fail criteria MUST carry the `[test-spec-change]` tag and a separate Engineer approval (not combined with the approval of the code fix).
- Isolation of the judge model: the production model ≠ the judge model.
- A trending test-fitting drift-rate metric.

Recovery: Restore the old criteria; perform a root-cause analysis — why the AI agent chose greening over a fix; update the prompt / system instructions.

3. The 14 AI risks (brief summary)

Full descriptions, mitigations, and owners — in [reference/03-ai-risk-register](#). Here is an operational summary: id, name, severity, the main detection signal.

ID	Name	Severity	Detection signal
----	------	----------	------------------

AIR-01	Hallucination in AI-generated requirements	High	Hallucination Rate metric > threshold; adversarial critic flags
AIR-02	Prompt injection via a client TZ	High	Suspicious pattern in imports; sandbox violation
AIR-03	Model drift / version change	Medium	diff regression on a model switch; baseline eval failure
AIR-04	Bias in AI requirement generation	Medium	Stakeholder map gaps; missing accessibility/locale considerations
AIR-05	Single-model failure (no diversity)	Medium	All artifacts with one <code>ai-provenance.model</code> ; no multi-model agreement
AIR-06	Test-fitting / greening tests	High	diff in TC pass/fail without a <code>[test-spec-change]</code> tag
AIR-07	Hallucinated citations	Medium-High	Citation validator hook fails
AIR-08	Adversarial inputs in client data	High	Application-level (out-of-scope for RENAR, tracked in SPEC-SEC)
AIR-09	Privacy leakage via AI logs	High	PII in the <code>tool_event</code> audit; redaction skip
AIR-10	Knowledge graph poisoning	Medium	Incorrect edges; circular dependencies in the graph
AIR-11	Reconciliation false-positive overload	Low-Medium	Findings/week trending up without real issues; high dismissal rate
AIR-12	Cost runaway (uncontrolled AI spend)	Medium	Project AI cost approaching the budget cap
AIR-13	A Stakeholder does not understand AI-generated requirements	Medium	Dispute rate at acceptance rising; long approval cycles
AIR-14	Vendor lock-in to a specific LLM provider	Medium	All prompts work only on one provider

The risk matrix and review cadence — [reference/03 §5-§2](#).

3.5 Adversarial review (procedure)

Informative. An operational procedure for WC-13; normative requirements — [standard/09 §9.4](#), [standard/13 §13.2 \(RENAR-5\)](#).

When mandatory (normative): adversarial review is QG-0 for RENAR-5 ([§11.8.1](#)); for `SPEC-SEC` / `SPEC-AI` — an external reviewer at QG-0 ([§5](#)); declared-stricter MAY broaden the scope ([standard/00 §0.6](#)).

Step	Actor	Artifact	Exit criterion
------	-------	----------	----------------

1. Scope	Architect	A list of TCs + the related SR/SPEC	Each TC norm of the approved version in scope has a <code>tc-type</code> and <code>verifies[]</code>
2. Critic pass	AI critic (a separate model/prompt)	A findings log with id, severity, and a reference to the TC/SR	Findings are traceable to a concrete clause §9.x; no "generic" recommendations
3. Triage	Architect + RE Engineer	Disposition: fix / accept / reject	Each finding has an owner + rationale; dismissal without rationale is forbidden (see §4.6)
4. Re-run	AI agent or human	Updated TCs + diff	QG-2 pre-condition: <code>passing-tests / total-tests</code> for the scope (§9.10)
5. Audit trail	substrate (V1)	A commit/change unit with the <code>adversarial-review</code> tag	provenance: model id, prompt version, findings hash (§10.13)

Approval discipline: the "100%" metrics in §9 are a **target at QG-2**, not a guarantee of product quality. AI-risk severity comes from [reference/03](#), not from an editorial override.

Agent panel (no human reviewers): an informative procedure — [§3.5](#) (steps 1–5); the rubric and severity — [reference/03](#).

4. Organizational failure patterns

These problems are not caught by substrate mechanisms — they are behavioral patterns of teams. They typically appear 2–6 months after adopting RENAR.

4.1 ACTZ as a formality

Symptom: The TZ clarification protocol is signed without being read. The questions were put to the client as the agent's raw output — a three-page list from which the client cannot tell what they are signing off on. The backward section of the ADAPT is empty or contains yes/no answers without context.

Sign: An ACTZ signed < 24 hours after the draft was generated; the rate of disputed requirements at acceptance is rising.

Mitigation: What is put to the client is a **prepared** list of decisions, not the agent's raw output ([standard/07 §7.13.4](#)): the Architect aggregates and rephrases the questions in the language of obligations. The dual signature belongs to the ACTZ, and to it alone; the ADAPT is signed by the Architect alone (§7.5), and demanding a client signature under it is pointless — that fiction is precisely what produced "signed without reading". The backward section MUST contain ≥ 1 non-rhetorical question. Spot-check ACTZs and ADAPTs in I&A.

4.2 SPEC overload

Symptom: The team creates a SPEC for every task, even when SR + TR are sufficient. The SPEC catalog balloons; every PR updates 5+ SPECs.

Sign: The SPEC / SR ratio > 1.5 (the expected value is < 0.3 for projects of medium complexity).

Mitigation: A pre-review checklist: "is a SPEC needed for this change?" A SPEC is justified only when several SRs share a common constraint. See [standard/08-specifications.md §8.2](#) — when a SPEC is mandatory.

4.3 Hooks as an obstacle

Symptom: The team routinely bypasses the substrate hooks (`--no-verify` , timestamp manipulation, manual status edits).

Sign: The git log / substrate audit trail shows a frequency of bypass commits; QG-0/QG-2 pass in suspiciously short times.

Mitigation: The root cause is hooks that are too slow / too noisy / too strict. Do not "ban the bypass" — fix the hooks. Treat the bypass frequency as a trending metric — if it rises, run a retro with the team.

4.4 Drift detection without action

Symptom: The reconciliation hook generates drift findings, but no one acts on them. The findings backlog grows; old findings are ignored.

Sign: Findings older than 14 days > 30; resolution rate < 20% / week.

Mitigation: Each drift finding gets an owner and an SLA (resolve / accept / reject within N days). Unresolved findings past the SLA are escalated. Reconciliation without human ownership = noise.

4.5 Tracker as a parallel universe

Symptom: The team lives in Jira / Linear / ADO; the `.req` directory is updated once a week "for the record". The tracker is the real Source of Truth, RENAR is a formal artifact for the audit.

Sign: The diff of `.req` vs the tracker > 30% in any given week; commits to `.req` are rare and batched.

Mitigation: The Source of Truth must reside in the substrate, not be tracker-resident. The tracker is a derived view only. If the team cannot work without the tracker — the substrate must push *into* the tracker, not the other way around.

4.6 Critic burnout

Symptom: The AI critic (adversarial review) generates many findings; gradually the developer / Architect start ignoring its output. Findings are rejected without consideration.

Sign: The AI critic's dismissal rate > 80%; time-to-dismiss < 30 seconds per finding.

Mitigation: Tunable thresholds for the critic. If the false-positive ratio is high — recalibrate the prompt / model. The "critic finding → real issue" metric (the % of dismissed findings that later surfaced as a defect) — if it is 0%, the critic is useless.

4.7 Single-engineer dependence

Symptom: Only one Engineer on the project "understands RENAR". All QG-0 / QG-2 pass through them. If they go on vacation — the process stalls.

Sign: The bus factor of RENAR ownership = 1. The distribution of QG approvals is heavily skewed toward one person.

Mitigation: Paired onboarding (at least 2 Engineers on the project know RENAR). Rotation of the QG-approver role. Documentation of project conventions in `<project>.req/CONVENTIONS.md`.

4.8 Ad-hoc delta

Symptom: Requirement changes happen without a delta-TZ being filed — "let's just change SR-12 right in the repository".

Sign: Direct commits to `<system>.req/sr/*` without a corresponding delta-TZ; the `created-by-order` field is empty.

Mitigation: The substrate hook blocks mutation of existing requirements without a `delta-ref` in the commit metadata. All changes go through the delta-TZ workflow ([standard/07-adapt §7.6](#)).

4.9 TC abandonment

Symptom: TCs are created alongside the requirements, but then they are never run. `last-run` is older than N months; the coverage report shows "green" TCs that in reality have not run in half a year.

Sign: Median `last-run` age > 90 days; the TC count grows, the run count does not.

Mitigation: The substrate runs TCs automatically on a schedule (capability V4). An implementation without a `last-run` for N days is automatically marked stale; QG-2 blocks the verification entry until they are re-run.

4.10 A test born green

Symptom: A TC written by the same agent, in the same session, as the implementation — and green from birth. Formally there is coverage; in fact the test checks the code, not the requirement.

Sign: The TC's run history contains no red result at all; the TC's provenance coincides with the provenance of the implementation change unit.

Mitigation: Authorship isolation along three axes — time (the TC norm is approved in a version and the implementation admitted before the TR starts), author (the test agent ≠ the implementation agent), and change (a Pass/Fail edit only via `[test-spec-change]`), [standard/09 §9.18](#). The pinning run before implementation MUST be red; a green one there is a signal to investigate, not a success. The single exception is a TC of class `implementation-originated`, which by construction has no red history: there its teeth are proven by a killed mutant.

4.11 Acceptance that confirms itself

Symptom: The ATs were generated by an agent that was given access to the SRs and the ADAPT "for context" — or simply by the same model that wrote the implementation. Every AT is green, and acceptance at the client fails.

Sign: The AT's `generator` does not attest isolation; or the AT's `tz-version` lags behind the current revision of the effective TZ.

Mitigation: ATs are derived **only** from the effective TZ, by an isolated agent on a different model, with no access to the internal loop ([standard/09 §9.19.2](#)) — otherwise the acceptance level collapses into the verification level and the error of interpretation receives a false confirmation of conformance. ATs are

regenerated before **every** round of trials: a program derived at planning time will, by the end of a long engagement, be checking the system against a year-old contract.

5. Failure recovery playbook

What to do once the system is already broken. The sequence is common to all failure modes; the specifics depend on the class.

Step 1: Stop the bleeding

Find and halt the ongoing damage:

- Drift: freeze further changes in the affected area.
- AI risk: suspend AI generation for the affected class of artifacts.
- Organizational: take it to a retro / I&A — this is not a technical fix.

Step 2: Quantify

Measure the damage:

- How many artifacts are in a drift state?
- How many releases since the problem arose?
- Which SR / SPEC / TC are affected? (Capability V4 — coverage / drift report)

Step 3: Triage

Segment the damage into:

- **Critical** — already in production, affecting users. Hot-fix.
- **Active** — in the current PI, affecting ongoing work. Block PI exit.
- **Historical** — old artifacts, not actively used. Batch fix.

Step 4: Fix

For each class, the corresponding fix:

- Schema drift → roll back the frontmatter; RFC if the schema needs to be extended.
- Implementation drift → delta-TZ adopt OR roll back the code.
- TC drift → re-create the implementation against the current set version (`automation.set-version`).
- Test-fitting → revert the criteria; root-cause the AI agent.
- Organizational → process retro + the specific mitigations (§4).

Step 5: Prevent recurrence

- Strengthen detection (a lower threshold, a new metric).
- Add a mitigation to the processed artifact.
- Record lessons learned in the project decision log or in the ADAPT backward findings (category `scope` / `terminology`).

Step 6: Verify

After the fix — re-run QG-2 on the affected artifacts. Drift detection should show a clean state.

6. Negative: what this chapter does not cover

- **Security incidents** — breach response, forensics, regulatory notification. This is an organization-level security process, not RENAR scope.
- **AI red team / penetration testing** — a separate security workflow; RENAR only tracks that the corresponding SR / SPEC-SEC should exist.
- **Compliance breach response** — a violation of GDPR / FZ-152 / PCI-DSS requires a legal process with the DPO / regulator, not a technical recovery.
- **Production incidents** — outages, performance regressions. These are operational; see the SPEC-OPS runbook.
- **Stakeholder conflicts** — disputes at acceptance, scope disagreements. RENAR provides the audit trail (who approved what, when), but resolution is a human process.

7. Relationship to other materials on failure modes

Document	What is in it	When to read
reference/03-ai-risk-register	The full register of 14 AIR risks with mitigations	When planning an AI use case; when reviewing the eval strategy
standard/04-terms §4.11	The closed list of drift classes with normative definitions	When disputing the terminology of failure modes
05-safe-comparison §9	The RACI matrix — who is accountable for each activity	When investigating an organizational failure
reference/04-ai-style-guide	The style of AI provenance; the minimal contract for AI-generated artifacts	When diagnosing AIR-01 (hallucination), AIR-07 (citations)

8. Resolved decisions for v1.0

- **A set of recovery steps with no platform binding.** The sequence in §5 is universal in nature. The details of "how exactly to freeze changes" for `git` and a document store are in [03-tool-guide-git §3](#) and [04-document-store-substrate](#). The scope of the normative minimum is set right here, in chapter 7.
- **Tuning the critic event-driven.** Re-tuning the critic's prompt is performed when the drift / hallucination metrics breach their threshold ([§12.3.3](#)); the RENAR-5 level requires continuous evaluation ([§11.8.1](#)), so a regular "general review for no reason" is redundant. On a metric trigger — it is permitted.

8.1 Deferred to v1.1 (phase-8 backlog)

- **Numeric thresholds for the organizational patterns (§4).** Today only qualitative "signs" are given. A set of acceptable values will be needed once field data has accumulated. Owners: the RENAR standard team and the adopting organizations.
 - **A formal measurement of the "bus factor" for §4.7.** The supporting tooling is not fixed; a possible approach is a graph query over commit authors across the revision chain (a built-in **V6** combination at the substrate). Owners: the authors of tooling for specific storage environments.
 - **Calibration of the provisional metric thresholds for RENAR-4 / RENAR-5.** The target values in §12.4 are set as direction markers and are subject to refinement once data for the metric have accumulated from five independent organizations with a completed cycle at each; until then they apply as they stand, but exceeding an uncalibrated threshold does not by itself remove conformance. The condition is declared met only by the formal change procedure of the standard (§13.9) through the publication of a calibration report — not by an individual and not by an adopting organization.
-

08. Developer Guide

Example for the `git` substrate only. This chapter illustrates **one** possible scenario, using GitHub and subsystems split across repositories (`.req` and `.src` directories). **RENAR is not tied to any specific substrate implementation**; your hooks and pipelines may be arranged differently. The generally applicable chapters are §6 Requirements hierarchy, §8 Specifications, §10 Lifecycle and quality gates. For an alternative that does not use a VCS as the file base, see 04-document-store-substrate.md.

Fictional company AcmeCorp: the `acme-platform` platform and four subsystems — `acme-portal` , `acme-ai` , `acme-orchestrator` , `acme-site` . Replace the names in the examples with your own; references to the normative chapters are unaffected.

1. What you need to know before you start

Two repository types. Each subsystem has two separate repositories:

Repository	Suffix	What's inside	Who writes it
Requirements	<code>.req</code>	BR, SR — what the system must do	Architect, Tech Lead
Source code	<code>.src</code>	Code, tests, CI/CD	Developer

The split is deliberate: requirements are versioned independently of the code, have a different review cycle, and have different access rights.

Hierarchy: System (`acme-platform`) → Subsystem (`acme-portal` , `acme-ai` , `acme-orchestrator` , `acme-site`) → Module (if present). Each level has its own `.req` repository; upper-level requirements are parents for the requirements below them.

Three requirement levels:

```
BR — Business Requirement (why the business needs it)
├── SR — System Requirement (what the system does)
│   ├── TR — Task Requirement (the specifics for implementation)
│       └── these are the fields of your task in the tracker, not a file
```

TR is not a file — it is Goal + Acceptance Criteria in a tracker task.

2. Initial setup of the local environment

Step 1. Confirm with your Tech Lead which subsystem you are working on: `acme-portal` (the customer portal), `acme-ai` (the AI pipeline), `acme-orchestrator` (the orchestrator), `acme-site` (the public site).

Steps 2-3. Create the structure and clone the repositories:

```

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

# Required for everyone – the system-level parent requirements (read-only):
git clone git@github.com:acmecorp/acme-platform/acme-platform.req

# Required – your subsystem's repositories:
SUBSYSTEM=acme-portal # replace with your own
mkdir -p $SUBSYSTEM
git clone git@github.com:acmecorp/acme-platform/$SUBSYSTEM/$SUBSYSTEM.req
$SUBSYSTEM/$SUBSYSTEM.req
git clone git@github.com:acmecorp/acme-platform/$SUBSYSTEM/$SUBSYSTEM.src
$SUBSYSTEM/$SUBSYSTEM.src

# As needed – the requirements of adjacent subsystems (read-only):
mkdir -p acme-ai && git clone git@github.com:acmecorp/acme-platform/acme-ai/acme-
ai.req acme-ai/acme-ai.req

```

Step 4. Verify: `find ~/projects/acmecorp -maxdepth 4 -name ".git" -type d | sed 's|/.git||' | sort`. The expected result for an Acme Portal developer:

```

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

```

3. Folder structure on the local machine

The local structure **mirrors** the repository hierarchy in the hosting platform — the relative `../..` paths between repositories become predictable.

For the directory layout (the substrate layout) see guide/03 §2 and §4: the canonical repository scheme + pinning `.src` → `.req` via a submodule (capability V5). Here is the typical local placement for an IDE: several clones lying side by side, without a mandatory submodule in the working folder.

```

~/projects/acmecorp/acme-platform/
acme-platform.req/           # system requirements (read-only)
acme-portal/
  acme-portal.req/           # your subsystem's requirements
  acme-portal.src/           # your subsystem's code – this is where you work
acme-ai/{acme-ai.req,acme-ai.src}/      # cloned as needed
acme-orchestrator/{acme-orchestrator.req,acme-orchestrator.src}/
acme-site/{acme-site.req,acme-site.src}/

```

Rule: do not change the folder names — they match the project names in the git hosting platform. This lets scripts and CI work with the same paths everywhere.

4. Setting up the workspace in the IDE

VS Code Workspace. Create `<subsystem>.code-workspace` in the subsystem folder:

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

Open it: `code ~/projects/acmecorp/acme-platform/acme-portal/acme-portal.code-workspace`. The side panel shows all three repositories at once.

JetBrains (IntelliJ, WebStorm, PyCharm). Open each repository as a separate module via **File** → **Attach Project** or **Project Structure** → **Modules**.

5. Working with requirements

5.1 How to read. Before starting a task, read the SR (`acme-portal.req/sr/SR-NN-*.md`). In the SR frontmatter, find the `parent` field — a link to the parent BR:

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

Open the parent BR — this is the "why the functionality exists".

5.2 Tracing. The chain: `Task TASK-42 → SR-01 (acme-portal.req/sr/...) → BR-01 (acme-platform.req/br/...)`. If anything is unclear, move up the chain.

5.3 Changing a requirement. The developer opens an Issue in the `.req` repository describing the mismatch between the SR and the actual behavior. The Tech Lead / Architect makes the change:

```
cd acme-portal.req
git checkout -b change/TZ-2026-002-totp
# Edit the SR file; update version + updated in the frontmatter; update
source.tz-section to the delta-TZ section
git add sr/SR-01-auth.md
```

```
git commit -m "[delta:TZ-2026-002] SR-01 v1.2: add TOTP"
# Open an MR/PR in the git hosting platform
```

Forbidden: committing requirement changes directly to `main` without an MR/PR and review.

5.4 What you must not do. Change `acme-platform.req` without the Architect's agreement; change the requirements of adjacent subsystems (`acme-ai.req` , `acme-orchestrator.req`); delete requirement files (only move them to `status: deprecated`); create version files (`SR-01-v1.md` , `SR-01-v2.md`) — the history lives in `git log` .

6. Working with tasks

6.1 Before you take a task — all QG-0 Approval Gate items are satisfied ([reference/01 §2.4](#); pre-v1.0 legacy "Context Gate"):

- The Goal is formulated; Acceptance Criteria exist — concrete, testable, independent.
- There is at least one negative scenario in the AC.
- The task references an SR (a field in the tracker); a work type is assigned.
- If it touches security — `threat-surface` is declared.

If something is missing — **do not take the task into work**; go back to the Supervisor/Tech Lead.

6.2 Acceptance Criteria. Each AC is a separate test. Good AC: `POST /auth/login` returns 200 and a JWT token with valid credentials ; `POST /auth/login` returns 401 with an incorrect password (a negative scenario); `POST /auth/login` returns 422 if the email field is missing ; the JWT token expires after 24 hours .

Bad AC (do not take such a task): "Login should work correctly"; "Error handling should be robust"; "The system should be secure".

6.3 If an AC is unclear or contradicts the SR: read the SR + the parent BR; add a comment to the task with a concrete question; do not interpret silently — an AI interprets silently, which is exactly why clear requirements are needed.

7. The Git process

7.1 Working with code (`.src`). The standard feature-branch workflow:

```
cd acme-portal/acme-portal.src
git checkout main && git pull
git checkout -b feat/TASK-42-totp-auth
# ... work ...
git add src/auth/totp.py
git commit -m "feat(auth): implement TOTP authentication (TASK-42)"
# Open an MR/PR in the git hosting platform
```

Commit format: `<type>(<scope>): <description> (<TASK-ID>)` .

7.2 Working with requirements (`.req`). The branch for changing a requirement (see §5.3 above).

7.3 Linking the MR/PR to the task. In the MR/PR description, specify: `Closes #42 + Related SR: SR-01 (acme-portal.req)` . If the change touches several `.req` repositories — `Related: acmecorp/acme-platform/acme-platform.req#15` .

7.4 Branch naming:

What you're doing	Branch
New functionality	<code>feat/TASK-NN-slug</code>
Bug fix	<code>fix/TASK-NN-slug</code>
Changing a requirement	<code>change/TZ-YYYY-NNN-slug</code>
New requirement	<code>feat/BR-NN-slug</code> or <code>feat/SR-NN-slug</code>

8. Common scenarios

Scenario 1. A new task: open it in the tracker → check QG-0 → find the link to the SR → open the SR in `acme-portal.req/sr/` → read the SR + the parent BR (if context is needed) → run the TCs bound to the task (they are already frozen and **red** — that is normal: they were written before you and not by you, [standard/09 §9.18](#)) → create the `feat/TASK-NN-slug` branch in `.src` → implement until the TCs turn green → MR/PR → review → merge.

***Why the test is already written, and not written by you.** A test born next to the code checks the code, not the requirement. So the TC is frozen before the task starts and is written by a different agent or engineer. If while implementing you added an internal technical detail that is not in the requirements (a defensive check, logging, an edge-case branch), it can be legalized through the class `source:`*

*`implementation-originated` ([standard/06 §6.13](#)): it needs provenance, explicit approval by a human supervisor, a covering test before the merge, and a **killed mutant** (such a test has no red history — it is born green). Client-observable behavior is not legalized this way: a screen, a field, a message, a deadline — those go through the contractual loop only.*

Scenario 2. A mismatch between the SR and the task's requirement: do NOT do anything silently → add a comment to the task ("SR-01 §4 describes X, the task requires Y — a contradiction, please clarify") → wait for the Supervisor/Architect's answer → do not take the task into work until it is resolved.

Scenario 3. Understanding where a requirement came from. In the `.req` repository: `git log - sr/SR-01-auth.md` (the change history) or `git show <commit-hash>` (the details of a specific change). Or, in the file's frontmatter, find the `source` field — a link to the TZ and its section.

Scenario 4. Integration (Acme Portal → Acme AI). Clone the requirements of the adjacent subsystem and add them to the workspace; open `acme-platform.req/specs/int/SPEC-INT-01-acme-portal-acme-ai.md` — the contract is described there. Integration contracts in canonical form are `SPEC-INT-NN` ([standard/08 §8.5.4](#)), not the legacy `INT-SR` . The subsystem's SR references them via `constrained-by: [SPEC-INT-NN]` . A SPEC-INT always lives in `acme-platform.req/specs/int/` — never inside the subsystems.

Scenario 5. Delta-TZ. This is the work of the Architect/Tech Lead, but for the developer: wait for the MR with the updated SRs to merge → `git pull` in `.req` → read `tz/TZ-YYYY-NNN-delta-N.md` (the delta-TZ text) → check whether the changes affect your current tasks → if so, update or close them in agreement with the Supervisor.

Scenario 6. Updating the local requirements repositories:

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

9. Access rights — who can change what

Repository	Developer	Tech Lead	Architect
<code>acme-platform.req</code> (system BR/SR)	Read-only	Read-only	Write via MR
<code>acme-portal.req</code> (subsystem SR)	Read-only	Write via MR	Write via MR
<code>acme-portal.src</code> (code)	Write via MR	Write + review	—
<code>acme-ai.req</code> (another subsystem)	Read-only	Read-only	—

If you need to change a requirement — open an Issue in the `.req` repository; do not commit directly.

10. Quick reference

Where to look when something is unclear:

Question	Where
What should the system do?	<code>acme-portal.req/sr/SR-NN-*.md</code>
Why does the business need it?	<code>acme-platform.req/br/BR-NN-*.md</code>
Where did this requirement come from?	frontmatter <code>source</code> → TZ
How did the requirement change?	<code>git log -- sr/SR-NN-*.md</code>
How does the integration with Acme AI work?	<code>acme-platform.req/specs/int/SPEC-INT-01-*.md</code>
What changed with the latest TZ?	<code>acme-platform.req/tz/TZ-YYYY-NNN-delta-N.md</code>

Checklist before creating an MR: the code covers all the ACs from the task; there are tests for the negative scenarios; the MR/PR contains a `Closes #NN` reference; if behavior changed — an Issue was opened in `.req` to update the SR.

Git commands (the most frequent):

```

git -C <portal.req> pull                                # update the requirements
git -C <portal.req> log -- sr/SR-01-auth.md              # the history of a
specific SR
grep -r "id: SR-01" ~/projects/acmecorp/ --include="*.md" # find a requirement
by ID
grep -r "SR-01" ~/projects/acmecorp/ --include="*.md"    # all tasks
referencing SR-01

```

Glossary: BR — Business Requirement; SR — System Requirement; TR — Goal + AC in the tracker; AC — Acceptance Criteria; QG-0 — Approval Gate (canonical v1.0; pre-v1.0 legacy "Context Gate"); **.req** — the git repository with the subsystem's requirements; **.src** — the git repository with the code; SPEC-INT — an integration specification (canonical v1.0; pre-v1.0 **INT-SR**); delta-TZ — an addition to the TZ on a repeat order; deprecated — an outdated requirement (not deleted, only flagged).

09. Examples library

Eight scenarios (E1–E8): six in-doc examples (E3–E8) plus two walkthroughs (E1–E2). Each in-doc example follows the chain **TZ** → **ADAPT** → **BR/SR** → **SPEC** → **TC** → **QG** in full or in part.

#	Scenario	Document	Audience	Focus
E1	Email/password sign-up (minimal)	00-quickstart	Beginner	30 min, minimal artifacts
E2	Login + profile + permissions (full)	01-walkthrough	Tech Lead, QA	Full lifecycle, AI generation of TC
E3	Personal-data export (GDPR / FZ-152)	§2	Legal, PM, Architect	Compliance, SPEC-DATA/SEC
E4	Webhook idempotency (REST API)	§3	Backend, Architect	<code>tc-type: contract</code> , SPEC-API
E5	RAG assistant (SPEC-AI eval)	§4	AI engineer	<code>tc-type: eval</code> , judge isolation
E6	Delta-TZ: scope dispute	§5	PM, Client, Architect	ADAPT backward, immutable TZ
E7	A subsystem as a standalone product	§6	Architect	The <code>implements[]</code> edge, §6.8.2
E8	The interface part of E3: an example to the end	§7	UI/UX, Architect, QA	Filled SR / SPEC-UI / ux - TC bodies

2. E3 — Personal-data export (GDPR Art. 15 / FZ-152)

Context: the SaaS "AcmeCRM" stores customer PII. The regulator and the contract require a machine-readable export to be delivered on a data-subject request within ≤ 30 days.

2.1 TZ fragment (immutable)

TZ-2026-002 – Data subject export

§3.1 A data subject can request a full export of their PII via the UI "Privacy → Export my data".

§4.2 Format: JSON + CSV bundle, zip, SHA-256 checksum.

§4.3 SLA: the download link is ready within ≤ 72 hours of verified identity.

§4.4 The export includes: profile, activity log for 24 months, marketing consents.

§4.5 The request is logged; a repeat export – no more than once every 30 days without an administrator override.

2.2 ADAPT (abridged)

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

Forward §3: the export is asynchronous (job queue); identity — via the existing MFA flow.

Backward (resolved):

#	Finding	Resolution
B-01	"24 months of activity" — calendar or rolling?	Rolling 730 days
B-02	Admin override — who approves?	Role <code>privacy-officer</code> + an audit log

2.3 BR + SR

```
# br/BR-02-data-subject-export.md
id: BR-02
type: BR
title: "The data subject receives a machine-readable PII export"
source: { adapt: ADAPT-002, adapt-section: "Forward §3" }
compliance:
  - { standard: GDPR, control: "Art. 15" }
  - { standard: "FZ-152", control: "art. 14" }
```

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

```
# sr/SR-09-export-delivery.md
id: SR-09
type: SR
parent: { id: BR-02 }
title: "The user downloads the ready bundle via a time-limited signed URL"
constrained-by: ["SPEC-API-04", "SPEC-SEC-03"]
```

2.4 SPEC (excerpts)

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

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

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

2.5 TC (pos/neg for SR-08)

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

```
---
id: TC-081
title: "Export rejected without an MFA-verified session"
type: TC
tc-type: security
automation: { status: automated, set-version: "1.0" }
verifies:
  - id: SR-08
negative: true
---
## When
POST /v1/privacy/export (session: password-only, no MFA)
## Then
- 403; job not created; security audit event
```

Analogous pairs for SR-09 (signed URL valid / expired).

2.6 Quality gates

Quality gate	Evidence
QG-ADAPT-approve (by meaning, near the "architecture gate": QG-3)	ADAPT-002 in <code>approved</code> , backward findings closed

2.7 What next

- The full scenario under `git` — [03-tool-guide-git](#)
- Compliance mapping — [06-compliance](#)
- Conformance manifest — [reference/08](#)
- The interface part of this same example, carried through to filled bodies — [§7 E8](#)

3. E4 — Webhook idempotency (SPEC-API)

Context: a payment provider sends `POST /webhooks/payment` with an `Idempotency-Key` . Duplicates MUST NOT create a double charge.

TZ (fragment): "A repeat webhook with the same key within 24h returns the same `payment_id` , HTTP 200."

SR + SPEC:

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

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

TC (contract pair):

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

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

Neg: a second key with a different body → 409 Conflict.

Quality gate QG-2 : both `TC` s pass; the `set-version` in `last-run` matches (`1.0`).

4. E5 — RAG assistant (SPEC-AI)

Context: an in-app chat answers from the knowledge base; eval against a golden Q&A set.

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

TC eval (judge ≠ production):

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

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

See [standard/09 §9.6.2](#), [guide/07 §3.5](#).

5. E6 — Delta-TZ: scope dispute

Context: after the signed TZ the client asks to "add PDF export" — outside the scope of ADAPT-002.

The correct path to the Source of Truth (SoT):

1. Do **not** edit the immutable TZ and do **not** silently extend the SR.
2. Draw up a **delta-TZ**; the adversarial reviewer reviews it and issues a verdict (§7.6.1). On a "findings present" verdict a delta-ADAPT `ADAPT-002-delta-1` is created (forward + backward); on "no findings" no delta-ADAPT is created and derived artifacts reference the delta-TZ via `source.tz-section` .
3. Backward finding: "PDF export was not part of Forward §3 of ADAPT-002."
4. The client's decision on the finding is drawn up as an **ACTZ** — a TZ clarification protocol with a dual signature ([standard/07 §7.13](#)); the finding receives `decided-in: ACTZ-003 §1` .
5. The delta-ADAPT is approved by the Architect's signature (§7.5) → new SR/SPEC/TC. The effective TZ is recomputed, and the ATs are regenerated from the new revision.

Anti-pattern: a direct commit to `sr/SR-08` without a `delta-ref` — drift class 4.11.6 ([standard/04 §4.11](#)).

See [standard/07 §7.6](#), [guide/02-transition-guide](#).

6. E7 — A subsystem as a standalone product (`implements -edge`)

Context: the platform `acme` (a system) consists of the subsystem `acme.notify` — a separate product with its own business owner (Notify Lead), its own team, and its own release cycle. Scenario §6.8.2 of the RENAR Standard.

6.1 Artifact hierarchy

```
acme (system)
├─ BR-01 (order intake with AI assistance)          level: system
├─ BR-05 (monitoring and alerts for the operations team) level: system
└─ acme.notify (subsystem, standalone product)
    └─ BR-01 (multichannel notification delivery)      level: subsystem
        implements:
            - id: BR-01      (elaborates "order intake": notifications on status
changes)
              scope: { system: acme }
            - id: BR-05      (elaborates "monitoring": notifications on SLA
breaches)
              scope: { system: acme }
            ↓
            SR-01..SR-12 (notify-internal requirements)
            SPEC-INT-01 (integration with acme via the message bus)
            SPEC-API-01 (REST API for notify clients)
```

`acme.notify.BR-01` is the root node of its own requirements tree (`parent` is absent, as for any BR). The link to the system is expressed by a **typed** `implements[]` edge, not a parent-edge.

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

```
---
id: BR-01
title: "Multichannel notification delivery"
type: BR
level: subsystem
scope:
  system: acme
  subsystem: acme.notify

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

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

# === implements-edge §6.8.2 ===
implements:
  - id: BR-01
```

```

scope:
  system: acme
  rationale: "ADAPT-NOTIFY-001 §2.1 – notifications on order status change"
- id: BR-05
  scope:
    system: acme
    rationale: "ADAPT-NOTIFY-001 §2.4 – SLA alerts for operations"

business-context:
  stakeholder: "Notify Lead"
  business-goal: "Timely delivery of notifications to the end-customer and the
operations team"
---

# BR-01: Multichannel notification delivery

The notify subsystem delivers notifications ...

```

6.3 Machine-readable trace chain

```

TC-NOTIFY-15
→ verifies SR-NOTIFY-08 (rate-limit for the email channel)
  |
  |— parent:   acme.notify.BR-01 v1.2
  |           |— implements: acme.BR-01 v3.0, acme.BR-05 v1.4
  |                               (typed cross-level edge)
  |— source.adapt: ADAPT-NOTIFY-001 §Forward §2.2
  |— constrained-by: SPEC-API-01, SPEC-OPS-01   (scope: acme.notify)

```

On an audit, the full chain is reconstructed from TC-NOTIFY-15: SR → subsystem BR → system BR. Before v1.0 (without the `implements` -edge) the chain broke off at `acme.notify.BR-01` and continued only through the prose of the "Context" section.

6.4 Gates on the substrate side

When `acme.notify.BR-01` is approved, the substrate-native hook checks (see [standard/10 §10.11.1](#)):

1. `acme.BR-01` and `acme.BR-05` exist in the substrate by `id + scope.system` .
2. Both target BRs belong to the approved set version of their system.
3. The chain `acme.notify.BR-01 → acme.BR-01 → ...` does not form a cycle.
4. `acme.notify.BR-01.level = subsystem` (rule: `implements[]` does not apply at `level: system`).

If any check fails, the approval is blocked. Behavior when `acme.BR-01 = deprecated` : a warning, not fatal (the Architect decides: update `implements` to the current BR or mark `acme.notify.BR-01` as requiring review).

6.5 Evolution "module → subsystem" via `implements`

When a module acquires a business owner ([standard/06 §6.9.1](#)):

1. The business owner is captured via an ADAPT backward finding (category `scope`).
2. After the delta-ADAPT is approved, the module is promoted to a subsystem; a subsystem BR is created.
3. **New step (v1.0+)**: the subsystem BR declares `implements[]` on the applicable system BRs. Without this step, the §6.8.2 scenario carries an absence of traceability incompatible with v1.1 conformance.

Anti-pattern: creating `acme.notify.BR-01` with `level: subsystem` but without `implements[]` — formally permitted on v1.0 (recommended), but non-conformant on v1.1+. If the parent system has an approved BR, the absence of `implements[]` MUST be **explicitly justified** in the "Context" section of the BR with a reference to ADAPT§.

See [standard/06 §6.5.2](#), [§6.8.2](#), [§6.10.3](#), [standard/13 §13.3.8](#).

7. E8 — The interface part of E3: an example carried through to the end

Informative section. *The example is not normative text and creates no obligations. Where it diverges from the chapters of the standard, the chapters prevail. The normative requirements the example follows are named by reference in every subsection.*

Why it is here. The other examples in the library stop at frontmatter and excerpts: they show the **shell** of an artifact — identifiers, statuses, graph edges. But the agent deriving the implementation reads not the shell but the **statement**. E8 continues E3 ([§2](#)) with an interface part and carries through to the end three bodies of which the corpus held not one: an SR body ([standard/06 §6.6.3](#)), a SPEC-UI body ([standard/08 §8.5.6](#)), and a pair of `ux` -TC ([standard/09 §9.6.1](#)).

What it proves — and what it does not. The boundary is better drawn up front, before reading. The example tests the **form** of the mandatory body: that the sections of §8.4.1 and §8.5.6 fill in meaningfully, that the requirement to "cover every action available through the interface" is operationalizable, that a `ux` -TC is written per §9.6.1 without inventing fields. The example does **not** test **volume**: it holds three screens, and on three screens the conclusion "the all-screens requirement is affordable on a real system" cannot be earned.

7.1 The boundary of "all screens" runs along the Scope of the specification

One particular of §8.5.6 surfaces only when one tries to write the body, and it is worth naming before the example.

The wording of the SPEC-UI mandatory body requires describing "all screens of the system". Read literally, it would mean that one SPEC-UI must describe the interface of the entire system — from sign-in to administration. That is not how it works and not how it was intended: the boundary of what is described is drawn by the **Scope section** of the same mandatory body ([standard/08 §8.4.1](#), item 2), while "all screens" means **completeness inside the declared Scope, with no selection by significance**.

Both halves matter:

- **Scope may be narrowed** — it is part of the mandatory body and is declared explicitly, so what falls outside it is not lost silently but visible as a boundary.
- **Inside the Scope nothing may be selected out** — this is precisely what the "Coverage completeness" clause of §8.4.1 prohibits: not "describe the important export screens" but "describe all the export screens".

Below, the Scope is declared as the personal-data export interface; it holds three screens, and all three are described.

Completeness across the system — the screen list. Scope and §8.4.1 give completeness inside a document; that the sum of all SPEC-UIs covers the whole interface is ensured by the screen list in the mandatory body of `SPEC-ARCH-01` of the E3 system and the coverage rule ([standard/08 §8.5.6.1](#)).

`SPEC-ARCH-01` was not shown in §2.4; here is the part of its list that concerns the "Privacy" section:

Code	Screen	Source	Covering SPEC-UI
S-01	Export request	<code>SR-10</code>	<code>SPEC-UI-02</code>
S-02	Export in preparation	<code>SR-10</code>	<code>SPEC-UI-02</code>
S-03	Request refused	<code>SR-10</code>	<code>SPEC-UI-02</code>
S-04	Second-factor identity confirmation	<code>SR-08</code>	<code>SPEC-UI-01</code>
S-05	Granting an override (the <code>privacy-officer</code> role)	<code>SPEC-ARCH-01</code> , the "privacy-officer workspace" container	—

The last column is not part of the list but the outcome of the check at the approval of the set version: `SPEC-ARCH` keeps the list, every `SPEC-UI` declares its coverage in `screens[ ]`. `S-05` is in the list and no `SPEC-UI` covers it — a set version with such a list **is not approved** ([standard/10 §10.7.2](#)) until the screen is described or removed from the list by a decision visible in the version record. The source of the list is the description only: `SR-08` and `SR-10` name the screens, `SPEC-ARCH-01` names the workspace container; a route found in the implementation and absent from the list is a source-of-truth drift and a backward finding, not grounds to extend the list after the fact.

7.2 SR-10 — a filled body (§6.6.3)

Continues `SR-08` from §2.3: that one describes the creation of the export job on the service side, `SR-10` — on the interface side.

```
# sr/SR-10-export-request-ui.md
id: SR-10
type: SR
parent: { id: BR-02 }
title: "A data subject initiates an export through the privacy interface"
source: { adapt: ADAPT-002, adapt-section: "Forward §3", tz-section: "§3.1" }
constrained-by: ["SPEC-UI-02", "SPEC-SEC-03"]
```

Requirement

WHEN the data subject confirms the request on the "Privacy → Export my data" screen, the system MUST create an export job and take the user to the status screen of that job.

*The form is event-driven (standard/06 §6.6.3.1): the condition marker WHEN stands as the first word.
The statement is one: creating the job and moving to the status screen are a single observable outcome of one action, not two independent requirements.*

Behavior

The request screen is available to the data subject in the "Privacy" section and shows the composition of the export before it is ordered: the profile, the activity log for 730 calendar days, and marketing consents (TZ §4.4 in §2.1, clarification B-01 from ADAPT-002).

WHILE the subject's identity is not confirmed by a second factor, the system MUST keep the "Order export" action unavailable and show the reason for unavailability next to it. Confirmation is performed by the existing second-factor mechanism; no separate mechanism is introduced for export.

IF less than 30 days have passed since the previous export and no override has been granted, THEN the system MUST refuse the order. The date from which an order becomes possible again is named to the subject on the refusal screen (TZ §4.5 in §2.1).

After the job is created, the system shows the preparation status and stays on the status screen until a download link appears or until the 72-hour preparation deadline expires (TZ §4.3 in §2.1). Delivery of the finished archive is regulated separately — SR-09 .

Every order, every refusal, and every override is written to the audit log (TZ §4.5 in §2.1, clarification B-02).

Constraints

Preparation deadline — no more than 72 hours from the moment of identity confirmation. Frequency — no more than one order per 30 days without an override. An override is granted by a participant in the privacy-officer role. Full constraints live in SPEC-SEC-03 .

Link to SPEC

SPEC-UI-02 governs the interface part: the set of screens, action coverage, the waiting and refusal states, accessibility, and language versions. SPEC-SEC-03 governs the frequency rule, the role granting an override, and the lifetime of the download link.

7.3 The five statement forms on one body of material

Standard/06 §6.6.3.1 introduces five forms. On the E3 material they look as follows — the table shows the form, not the full text of the requirements:

Case	Statement
Ubiquitous	The system MUST write every export order to the audit log.
State-driven	WHILE the subject's identity is not confirmed by a second factor, the system MUST keep the "Order export" action unavailable.
Event-driven	WHEN the data subject confirms the request, the system MUST create an export job.

Unwanted behavior	IF less than 30 days have passed since the previous export, THEN the system MUST refuse the order.
Optional feature	WHERE a granted override is present, the system MUST accept the order earlier than 30 days.

Three particulars of the form are visible in these same rows: WHERE marks presence, not place; THEN after IF is mandatory; one statement — one sentence, so that where one was tempted to write "refuse and name the date", the refusal and the message are split into two sentences, because a glued statement is verified indivisibly and a single TC would prove half of it.

7.4 SPEC-UI-02 — a filled mandatory body (§8.4.1 + §8.5.6)

```
# specs/ui/SPEC-UI-02-privacy-export.md
id: SPEC-UI-02
type: SPEC
spec-type: SPEC-UI
title: "Personal-data export interface"
source: { adapt: ADAPT-002, adapt-section: "Forward §3" }
depends-on: ["SPEC-SEC-03", "SPEC-API-04"]
screens: ["S-01", "S-02", "S-03"]      # codes from the SPEC-ARCH-01 screen list (§7.1)
ui-platform: web
target-users: [{ role: data-subject, persona: "ADAPT-002 §2.1" }]
design-system: "AcmeCRM Design Kit 4"
accessibility-level: WCAG-AA
i18n: required
mockup-links: [{ tool: figma, url: "<mockup link>", version: "v3" }]
baseline-images:
  - "ai-concepts/baselines/SPEC-UI-02-screen-01.png"
  - "ai-concepts/baselines/SPEC-UI-02-screen-02.png"
  - "ai-concepts/baselines/SPEC-UI-02-screen-03.png"
```

Purpose

The specification governs the interface through which a data subject orders an export of their personal data, watches it being prepared, and receives a refusal with an explainable reason. It gives the agent implementing the interface a single source of truth about the set of screens and about what is available on them; whatever is not described here has no source of truth and is not subject to guesswork ([standard/02 §2.3.3](#)).

Scope

In: the three screens of the "Privacy → Export my data" section — export request, preparation status, refusal; every action available to the data subject on those screens; the cross-cutting elements of those screens.

Out: the second-factor confirmation screen (S-04 of the list: the rule — `SPEC-SEC-03`, the screen — `SPEC-UI-01`; reused unchanged); the interface of the participant in the `privacy-officer` role

who grants the override (a separate specification, not covered in E8); the email notification that the archive is ready (not a screen).

Overall interface structure

"Privacy" is a tab in account settings. Export is one of its items; the export screens form a linear sequence with no branching: request → status, or request → refusal. A return to settings is available from any of the three screens. There are no pop-up dialogs in the section: refusal and status are full screens, because both are addressable by link and both must survive a page reload.

Screens

There are three screens; all three are described.

Code	Screen	When it is shown
S-01	Export request	The subject opened "Export my data"; no active job exists
S-02	Export in preparation	A job is created and unfinished, or finished and the link is still alive
S-03	Request refused	The order was refused by the frequency rule or by unconfirmed identity

S-01 "Export request". Contains: the section heading; the list of what will go into the archive — profile, activity log for 730 days, mailing consents; a line about the preparation deadline of up to 72 hours; a line about the frequency of no more than once per 30 days; an identity-confirmation marker with its current state; the "Order export" action; the "Back to settings" action.

S-02 "Export in preparation". Contains: the preparation state in words, without a completion percentage — the share is unknown on the interface side; the job creation time; the preparation deadline; the "Refresh status" action; the "Cancel export" action; the "Back to settings" action. Once the link appears, the same screen gains the "Download archive" action and the link lifetime.

S-03 "Request refused". Contains: the reason for refusal in plain language; the date from which an order becomes possible again — for a refusal by frequency; the "Request an override" action; the "Back to settings" action. For a refusal by unconfirmed identity, the "Confirm identity" action is shown instead of the date.

User journeys and action coverage

Coverage is checked by name: every action of every screen is named below at least once ([standard/08 §8.5.6](#) — "covering every user action available through the interface").

Journey	Screens	Actions covered
J-1. Ordinary order	S-01 → S-02	Expanding the archive composition list, "Order export", "Refresh status", "Download archive"
J-2. Order before identity confirmation	S-01 → S-03 → S-01	"Confirm identity"
J-3. Repeat order before the deadline	S-01 → S-03	"Request an override"
J-4. Abandoning a started export	S-02 → S-01	"Cancel export"

J-5. Leaving the section	S-01, S-02, S-03	"Back to settings" from each of the three screens
--------------------------	------------------	---

Reconciliation by count: there are eight actions across the three screens — expanding the archive composition list, "Order export", "Back to settings", "Refresh status", "Cancel export", "Download archive", "Request an override", "Confirm identity". All eight are named in journeys J-1...J-5. There are no uncovered actions.

Cross-cutting elements

Access rights. All three screens are available only to the account owner. A participant in the **privacy-officer** role does not see these screens and cannot order an export on the subject's behalf; their interface is out of Scope.

Notifications. The interface does not itself notify of readiness: the notification goes by email. S-02 says so explicitly, so the subject does not keep the tab open.

Error states. A refusal by the rules is S-03, a full screen. A connection failure and unavailability of the export service are shown on the screen where they occurred, as a message with a "Retry" action; they do not move the user to S-03, because the order was not refused but not delivered.

Empty states. S-01 has no empty state — the composition list is never empty. On S-02 an empty state is impossible by construction: the screen is shown only when a job exists.

Tone and style

The subject is addressed formally. The reason for refusal is named directly and without apology: "Export is available once every 30 days. Your next order — from 12 June". Service words — "job", "archive", "activity log" — are shown to the user as such and are not replaced by internal identifiers.

Accessibility

The level is WCAG-AA. What matters for these three screens: a state change on S-02 is announced to the screen reader as a live region, otherwise the subject never learns the archive is ready; the reason the "Order export" action is unavailable is associated with the action programmatically, not merely placed next to it; a refusal on S-03 is not conveyed by color alone; the keyboard traversal order matches the reading order on all three screens.

Language versions

Russian and English are mandatory. The date of a possible repeat order is rendered in the locale format, not in one shape for both versions. The length of the refusal-reason line is not bounded by the button width: the English wording is longer than the Russian one, and the screen must wrap it rather than truncate it.

Link to requirements

SR-10 — ordering an export through the interface (**constrained-by**). **SR-09** — delivery of the finished archive, whose interface part is limited to the "Download archive" action on S-02. **BR-02** — the business requirement from which both are derived.

Link to other SPEC

SPEC-SEC-03 — the frequency rule, the role granting an override, the link lifetime; the interface displays these rules but does not define them. **SPEC-API-04** — the source of the job states displayed

on S-02.

Verification

TC-090 and TC-091 (§7.5). Accessibility is verified by separate TC of type `accessibility`, language versions by TC of type `i18n`; they are not expanded in E8.

7.5 A pair of `ux` -TC (§9.6.1)

The two-layer structure of §9.6.1: the intent layer is executed by the driving agent, the perceptual-check layer by the judging model. The mandatory frontmatter extension is `judge.vendor`, `judge.model`, `baseline.artifact`, `baseline.perceptual-diff-threshold`. Body sections follow [standard/09 §9.4](#), including the mandatory "Out of scope".

```
---
id: TC-090
title: "The subject orders an export and sees the preparation screen"
type: TC
tc-type: ux
automation: { status: automated, set-version: "1.0" }
verifies:
  - id: SR-10
    statement: 1 # SR-10#1 – "Requirement", event-driven form
negative: false
judge: { vendor: google, model: "gemini-2.5-pro" }
baseline:
  artifact: "ai-concepts/baselines/SPEC-UI-02-screen-02.png"
  perceptual-diff-threshold: 0.03
---
```

Context. `SR-10`, the "Requirement" section: "WHEN the data subject confirms the request ... the system MUST create an export job and take the user to the status screen of that job". Screen S-02 per `SPEC-UI-02`.

Preconditions. A subject account with a confirmed second factor; no active export jobs; the previous export was more than 30 days ago. The state is prepared by the seed mechanism.

Steps. Intent, not selectors: the data subject wants to obtain an archive of their data after entering the privacy section.

Pass criterion. The rendered state shows the preparation screen: the state is named in words, the job creation time and the preparation deadline are shown, the refresh-status action is available. The perceptual difference from the baseline does not exceed 0.03.

Fail criterion. Observable signs of violation: still on the request screen; completion shown as a percentage; the deadline missing or shown as "unknown"; a download link presented before the archive is ready; two jobs created on a single confirmation.

Postconditions. Exactly one export job in the preparation state; an audit log entry created. The job is removed by the cleanup mechanism after the run.

Out of scope. Delivery of the finished archive and the link lifetime are not verified — covered by the TC pair for `SR-09` . Refusal by the frequency rule is not verified — covered by `TC-091` . Screen accessibility is not verified — covered by TC of type `accessibility` .

```
---
id: TC-091
title: "A repeat order before 30 days is refused with the date named"
type: TC
tc-type: ux
automation: { status: automated, set-version: "1.0" }
verifies:
  - id: SR-10
    statement: 5
    # SR-10#5 - "Behaviour", refusal by the
    frequency rule
  negative: true
judge: { vendor: google, model: "gemini-2.5-pro" }
baseline:
  artifact: "ai-concepts/baselines/SPEC-UI-02-screen-03.png"
  perceptual-diff-threshold: 0.03
---
```

Context. `SR-10` , the "Behavior" section: "IF less than 30 days have passed since the previous export and no override has been granted, THEN the system MUST refuse the order". Screen S-03 per `SPEC-UI-02` .

Preconditions. A subject account with a confirmed second factor; the previous export finished 5 days ago; no override was granted.

Steps. The data subject wants to order an export again shortly after receiving an archive.

Pass criterion. The rendered state shows the refusal screen: the reason is named in plain language, the date of the next possible order is given, the override-request action is available. The perceptual difference from the baseline does not exceed 0.03.

Fail criterion. Observable signs of violation: an export job created against the rule; the refusal shown without the next-order date or with a date in the past; the refusal conveyed by color alone, without text; the refusal shown as a pop-up instead of a screen; the override-request action missing; the reason rendered as a raw service error message.

Postconditions. No new export jobs; an audit log entry about the refusal is created.

Out of scope. Granting the override by a participant in the `privacy-officer` role is not verified — their interface is out of `SPEC-UI-02` Scope. Refusal by unconfirmed identity is not verified — a separate TC of the same form pair.

Baselines. Both TC are compared against the baseline images from the specification's `baseline-images` . Replacing a baseline goes through the approval mechanism with the `[baseline-update]` tag ([standard/09 §9.13](#)); automatic update is prohibited, otherwise a changed interface declares itself correct.

7.6 SPEC-UC-01 — use case (§8.5.12)

The scenario stitches `SR-10` , `SPEC-UI-02` and `SPEC-API-01` ; the executor is a person, so every step is covered by a `ux` TC and not by a `system` TC bypassing the interface ([standard/09 §9.8](#)).

```
---
id: SPEC-UC-01
title: "Ordering a personal-data export by the data subject"
type: SPEC-UC
spec-type: SPEC-UC
level: system
role: human
persona: "ADAPT-003 §Persona: data subject"
covers: [SR-10, SPEC-UI-02, SPEC-API-01]
entry: SPEC-UI-02
steps:
  - { n: 1, action: "Opens the export request screen", ref: SPEC-UI-02, expects:
    "Screen S-01 with the request form" }
  - { n: 2, action: "Confirms the request", ref: SR-10#1, expects: "An export job
    is created; transition to S-02" }
  - { n: 3, action: "Waits for preparation", ref: SPEC-UI-02, expects: "S-02
    shows the job state" }
  - { n: 4, action: "Gets a refusal when re-ordering within 30 days", ref: SR-
    10#5, expects: "S-03 with the reason for refusal" }
source: { adapt: "ADAPT-003", adapt-section: "Forward §4", tz-section: "§5.2" }
---

## Goal and executor
The data subject orders an export of their data and sees the job state.

## Preconditions
Identity confirmed; no previous export, or ≥ 30 days have passed.

## Main path
Steps 1-3 from the frontmatter; each references a statement of `SR-10` or a
screen of `SPEC-UI-02`.

## Alternative and negative branches
Step 4 – refusal by period (`SR-10`, "Behaviour"); refusal by unconfirmed
identity – outside the Scope (see `SPEC-UI-02`).

## Postconditions
The export job exists in the "in preparation" state, or the refusal is recorded
in the log.

## Covering TCs
`TC-090` (steps 1-3, positive), `TC-091` (step 4, negative); both `tc-type: ux`.
```

A step without `ref` violates structural completeness (§8.5.12.1): a set version with such a scenario is not approved.

7.7 What this example does not prove

Stated explicitly, so the example is not read as more than it is.

- **The affordability of "all screens" on a real system.** There are three screens. On three screens the form of the mandatory body was tested, not its volume. A product with fifty screens will carry a different cost, and this example does not measure it.
- **The fitness of the statement form against accumulated description.** The five forms of §6.6.3.1 were applied to five statements written in the form from the outset. Converting an existing description into these forms is a different task, and the modality of the rule stays advisory precisely for that reason ([standard/06 §6.6.3.1](#)).
- **A machine check of screen-list coverage.** The table in §7.1 was compiled by hand; a static TC that reconciles `screens[ ]` of all SPEC-UIs against the `SPEC-ARCH-01` list is not written in the example.
- **The workability of the perceptual check.** `TC-090` and `TC-091` are written per §9.6.1 but were never run: there are no baseline images in the corpus, and no judging model was connected to the example. What was verified is that the fields of §9.6.1 fill in without inventing new ones — no more.

10. Migration to v1.0

Historical guide v0 → v1.0. The `requirement-version` field, artifact statuses and the `approved → verified` transitions below describe v1.0; in v1.1 they are replaced by the description set version (`set-version` , standard/10 §10.5) — the v1.0 → v1.1 migration guide ships with v1.1.

For teams that already managed requirements "their own way" or on early RENAR drafts. The goal is **no big-bang**: preserve immutable IDs, replace deprecated constructs, and issue a new conformance manifest. The normative basis — standard/04 §4.14, CHANGELOG §Migration.

Do not confuse this with 02-transition-guide — that covers a staged entry into RENAR-1..5 from scratch; here we deal with a **breaking rename** and schema alignment when moving to the current edition of the standard.

1. When this migration is needed

Situation	Action
A project with no RENAR, but with a TZ + Jira	First 02-transition-guide , not this document
Files with INT-SR , INT-TC , AIC , UIC , TS	This guide
<code>verifies[].version</code> without <code>requirement-version</code>	Update the TC frontmatter (reference/02 §8)
Manifest <code>renar-version: 0.1-draft</code> or absent	Issue a new manifest (reference/08)

2. Type-replacement table (closed list, v1.0)

Deprecated	Canonical	Migration step
INT-SR	SR + constrained-by: [SPEC-INT-N]	Rename the type; create / bind a SPEC-INT
INT-TC	TC + tc-type: contract	Add type: TC , tc-type: contract
AIC	SPEC-AI	Move the body into the SPEC-AI frontmatter
UIC	SPEC-UI	Move baselines into <code>specs/ui/baselines/</code>
TS	SPEC-ARCH or SPEC-OPS	By content (architecture vs ops runbook)
TM (module SR type)	SR + level: module	Drop the pseudo-type; set the level

<code>tc-type:</code> <code>acceptance</code>	<code>tc-type:</code> <code>business</code>	Rename the value. There are now two acceptances: a <code>business</code> TC measures the BR's business goal (the internal loop), while conformance to the contract is checked by <code>AT</code> (standard/09 §9.19) — one name for two different things has been removed (§9.5)
<code>approval.client-signature</code> on an ADAPT	The Architect's signature alone	Remove the client signature from the ADAPT; move the client's decisions into signed <code>ACTZ</code> s and set <code>decided-in</code> on the findings (standard/07 §7.5 , §7.13)
ADAPT state <code>client-ready</code>	<code>asked</code>	Removed from the state machine: putting questions to the client is the business of the ACTZ, not a state of the ADAPT (standard/10 §10.8)

Do not change IDs. If a filename contains a legacy prefix, a `legacy-id` field in the frontmatter is acceptable (informative traceability).

3. Step-by-step plan (1–2 sprints)

Phase A — inventory (1–2 days)

1. `grep` / search across the substrate: `INT-SR` , `INT-TC` , `AIC` , `UIC` , `type: system` (the erroneous TC type).
2. List the artifacts with broken `verifies` / missing `source.adapt` .
3. Record the current `RENAR-CONFORMANCE.yaml` (if any) as the baseline state.

Phase B — schema pass (3–5 days)

1. Batch-rename the types per the §2 table (one PR per type, or one atomic change-set per delta-TZ).
2. TC: `type: TC` , `tc-type` , `verifies[].requirement-version` , `last-run.requirement-version` .
3. SR: `constrained-by[]` on every SPEC in use.
4. BR: the adaptation source — `source.adapt` on the approved ADAPT, or `source.adversarial-review-ref` on the issued AR when the verdict is "no findings"; `source.tz-section` is always mandatory ([standard/07 §7.4.1](#)).

Phase C — validation (1–2 days)

1. Substrate hooks / CI: the frontmatter validator ([reference/02](#)).
2. Run the TCs; update `last-run` .
3. Self-assessment checklist — [reference/08 §2](#).

Phase D — manifest (1 day)

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

3. Signature of the Architect / Tech Lead (V6).

4. Common pitfalls

Mistake	Consequence	Fix
Renaming <code>SR-05</code> → <code>SR-05-v2</code>	V1 immutable-ID violation	<code>deprecated</code> + a new ID with <code>replaces</code>
Leaving <code>type: system</code> on a TC	Validator fail; KG drift	<code>type: TC</code> + <code>tc-type: system</code>
Migrating code before the <code>.req</code>	SoT inversion violated	First the SR/SPEC/TC approved, then the TR
Skipping the adversarial review "only for new TZs"	Non-conformant for legacy TZ	A retrospective adversarial review of every active TZ; an ADAPT only on a "findings present" verdict (standard/07 §7.4.1)

5. Rollback

The migration runs through the substrate history (V1). Rollback means reverting the change-set / PR, **not** deleting artifacts. Deprecated artifacts remain with `status: deprecated` for audit.

6. Related documents

Document	Why
02-transition-guide	RENAR-1..5 without schema breaking changes
09-worked-examples	Reference frontmatter after migration
reference/07	External ISO 29148 statement
standard/13 §13.7	Re-assessment after migration

RENAR Guide 1.0 — [renar.tech](#)

11. Working with the AI agent

This is not "manual mode." A common misconception: since RENAR requires human approvals, everything must be done by hand. The opposite is true — the bulk of the work (drafts of `BR / SR / SPEC / TC`, the initial interpretation of the TZ, the search for contradictions) is done by the agent. The human does not type artifacts; the human **approves** them at a few fixed points. This chapter shows how to load the standard into the agent, what commands to give, and what you get as output.

The normative delegation model — `standard/05 Roles` and the ADAPT approval points — `standard/07 §7`. The self-sufficient edition for the agent is `RENAR-AGENT-EN.md` (in the repository root and at `renar.tech`).

1. Who does what: the delegation map

The agent is an executor: it produces and maintains artifacts, but it does not own them and does not sign. The human is the owner and the one who approves. The boundary runs along fixed points:

Step	The agent does	The human approves
TZ import	Registers the TZ as an immutable document, captures the client's signature	— (the client signs when delivering the TZ)
Adversarial review	A critic agent on a different model issues the verdict "findings present" / "no findings"	The Architect records the verdict as evidence
ADAPT	Prepares a draft: forward interpretation + backward findings	The Architect's signature — and only theirs: the ADAPT is not put to the client
ACTZ	Prepares a draft TZ clarification protocol from the findings	Dual signature: client + contractor. Before it goes out, the Architect aggregates and rephrases the questions
Decomposition	Creates <code>BR</code> → <code>SR</code> → <code>SPEC</code> → <code>TR</code> with provenance	The Architect approves the transition to <code>approved</code> (QG-0)
Tests	Writes <code>TC</code> pairs (positive + negative) — before the implementation and not by the agent that writes it	The automated run records the result (QG-2)
Acceptance tests	An isolated agent on a different model derives <code>AT</code> from the effective TZ alone	The release gate: every <code>AT</code> green
Delivery	Presents the business outcome	The stakeholder accepts it (QG-4)

If human approval is not obtained, the artifact stays in `draft`, and the substrate blocks the transitions. The agent never declares itself the owner and never signs.

The boundary between the two loops. Everything shown to the client and signed is an obligation (ACTZ). Everything not shown is an interpretation (ADAPT). The agent **MUST** tell them apart: a question that requires a client decision goes into a draft ACTZ; it is not "filled in" inside the forward interpretation ([standard/07 §7.13](#)).

1.1 The roles of a vendor's production model ↔ RENAR roles

A vendor organization usually keeps a role list of its own. RENAR does not redefine it ([standard/05 §5.2.2](#)): it is enough to map every role to the closed list of §5 and to a signature zone. An example mapping for a production model with seven roles:

Vendor role	Zone of responsibility	RENAR role (§5.2.1)	Signs
Project manager	the order portfolio and capacity	Supervisor	—
Chief architect	the architecture portfolio, stacks, verification rules	Architect (organization-level norms; tightenings — declared-stricter , §5.7.2)	stack rules, standard solutions
Client manager	the interface with the client	the vendor's authorized person (§5.4)	vendor-signature of the ACTZ; arranges the stakeholder's signature
Order system architect	the architecture of the order, the description set	Architect — owner of ADAPT, SPEC, the set version (§5.3)	ADAPT, the set version, protocols, benches (SPEC-TEST)
Development supervising engineer	the agent fleet, the course of development	Supervisor; task acceptance — under the prohibition to combine execution and acceptance of one task (§5.5.5)	task acceptance
AI engineer	agent profiles, prompts	Supervisor (AI-agent profiles; ai-provenance , standard/04 §4.10.1)	—
Platform administrator	infrastructure	not a RENAR role — provides the substrate capabilities V1–V6 (standard/03)	—

The client is the Stakeholder ([client-signature](#)); the agent is the AI agent: it produces artifacts, owns none and signs nothing; the operator is the Supervisor. One person **MAY** combine roles, and that is named, not compensated; only the execution and the acceptance of one task and the two sides of the ACTZ signature are not combined.

2. How to load the standard into the agent

The agent does not need the entire normative corpus — for everyday work a single file is enough.

1. Download [RENAR-AGENT-EN.md](#) — the self-sufficient operational edition (≈400 lines): the artifact map, the preconditions of artifacts, ADAPT, the closed lists, the hard rules, the recording forms (frontmatter + body sections).

2. Place the file beside the agent — in the project context, the system prompt, or as a session attachment (depends on your tool).
3. Give the setup command: "Study this file — it is the RENAR standard. From now on do requirements engineering strictly by it."
4. Optionally add project context: the registry of systems and subsystems, a link to your artifact substrate, the naming convention.

Attach the full corpus (`standard/` , `reference/`) only when a formal dispute over the exact wording of a norm arises — for most tasks the operational edition is enough.

3. How to give commands

Commands to the agent are ordinary natural-language statements tied to the artifact the agent produces. Below are typical examples; substitute your own identifiers.

TZ import and review:

```
Here is the client TZ (file TZ-2026-001). Register it as an immutable document.
Then run an adversarial review: issue the verdict "findings present" or "no
findings",
with reasoning across the 7 categories (contradiction, gap, hidden assumption,
feasibility, regulatory, terminology, scope).
```

Creating an ADAPT (if there are findings):

```
The review found findings. Prepare an ADAPT draft: a forward interpretation by TZ
section plus backward findings. Do not fill in for the client – turn whatever is
unclear into a finding. The ADAPT is an internal document; it does not go to the
client.
```

The TZ clarification protocol (ACTZ):

```
Collect the findings that require a client decision into a draft ACTZ – "TZ
Clarification
Protocol No. 1". Phrase every clause in the language of obligations ("the domain
is
compared case-insensitively"), not of interpretation, and add the contractor's
proposal.
Status: draft – putting it to the client and signing it is my job.
```

Decomposition:

```
ACTZ-001 is signed by both parties and the ADAPT is approved. Set decided-in on
the
protocol clauses in every closed finding. Derive a BR (business goal) from the
```

```
ADAPT,  
then SR (system behavior), then SPEC of the needed types. Set source and parent  
on  
every artifact. Show the relationship graph before committing.
```

Tests:

```
For every normative statement in SR-03, write a TC pair: positive and negative.  
The Pass criterion must be binary and observable. There is no implementation yet,  
and  
there must not be: first we freeze the tests, then the code – and a different  
agent  
will write the code.
```

Acceptance tests (a separate session, a different model):

```
Here is the effective TZ: TZ-2026-001 plus the signed ACTZ-001 and ACTZ-002. That  
is all  
you have and all you will get – no requirements, no specifications, no tests, no  
code.  
Derive the ATs: acceptance tests per contract clause, with a verbatim quote of  
the clause  
next to the verification steps, in positive/negative pairs.
```

A good command names the **input** (which artifact), the **action** (what to produce), and the **expectation** (what to check). The agent handles the routine itself; step in where your decision is needed.

4. What you get as output

The agent returns not chat text but ready substrate artifacts — with machine-readable frontmatter and a body in fixed sections:

- **BR** — the business need: who, what, why; success criteria; context with a reference to an ADAPT.
- **SR** — system behavior: one requirement, behavior, constraints, link to **SPEC** .
- **SPEC-<TYPE>** — a specification of one of the twelve types, linked to the **SR** via **constrained-by[]** .
- **TC** — positive/negative pairs with **verifies[]** pointing at the verified artifact.
- **ACTZ** — a draft TZ clarification protocol: decision clauses in the language of obligations, **status: draft** until it is put to the client.
- **AT** — acceptance tests referencing the TZ and the ACTZs only, with a verbatim quote of the contract clause and the provenance of the isolated generator.

Every artifact has its **source** set (origin from the TZ or an ADAPT) and, for AI-generated ones, an **ai-provenance** block (model, prompt template, the fact of a human edit). Ready copy-paste skeletons for all forms are in [reference/12 Document templates](#).

The mark of good output: from any artifact a chain traces upward — to the `SR` , to the `BR` , to a TZ section. If the chain breaks, the artifact is not ready.

5. Points of human approval

Delegation to the agent is broad, but it has hard boundaries. A human is required in five places (normatively — [standard/10 Lifecycle and quality gates](#)):

1. **TZ signature** — the client signs the contractual input; the agent only registers it.
2. **The ACTZ dual signature** — the client and the contractor sign the TZ clarification protocol. This is the only place where an obligation to the client comes into being ([§7.13](#)).
3. **The Architect's signature on the ADAPT (QG-3)** — the Architect confirms that every finding is worked through (each carrying a `decided-in` on a clause of a signed ACTZ) and that the interpretation is feasible. The client does not sign the ADAPT: they never saw it ([§7.5](#)).
4. **QG-0 approval** — the Architect moves `BR / SR / SPEC` from `draft` to `approved` . Only after this is further decomposition allowed.
5. **Acceptance** — the release gate on the ATs (all green) and, optionally, QG-4 on the business outcome.

Between these points the agent acts on its own. An attempt to make the agent sign, or to declare it `Accountable` , breaks the role model.

6. Who writes the test, and why it MUST be red

This is the one part of the work where the agent is explicitly **forbidden** to do everything at once — and the prohibition is substantive, not ceremonial.

Authorship isolation ([standard/09 §9.18](#)). A test written by the same agent that writes the code — and especially one written **after** the code — honestly checks what was written. But what it ought to check is the requirement, not the code. So the isolation rests on three axes:

Axis	What it means in practice
Time	The <code>TC</code> is frozen in <code>ready</code> before the task starts. The substrate will not let a TR open while the tests are unfrozen
Author	The agent writing the <code>TC</code> is not the one writing the implementation: a different session or a different model
Change	An edit to the Pass/Fail criteria of a frozen test is approved by an engineer; silently "nudging the test to fit the code" is not allowed

Red history ([§9.18.2](#)). A test never observed red is not evidence. The pinning run happens **before** implementation, and the test **MUST** fail: the functionality under test does not exist yet. Green on the pinning run is not a success but a signal to investigate: either the test checks nothing, or the functionality is already there.

There is one exception — tests of class `implementation-originated` ([standard/06 §6.13](#)): an internal technical detail added by the agent during implementation (a defensive check, logging) that is not observable by the client. Such a test is written after the code and is born green — by construction it has no

red history. The compensation is mandatory: its teeth are proven by a **killed mutant**. And the class boundary is hard: client-observable behavior is not legalized through it — an obligation to the client cannot arise out of code.

Isolation of the AT generator (§9.19.2). The acceptance tests are derived by a separate agent on a different model, whose input is **only** the effective TZ. Access to the ADAPT, the **BR / SR / SPEC**, the **TC**s, and the code is forbidden — not out of hygiene, but because an AT is a simulation of the client's gaze. An agent that has seen the ADAPT will reproduce the same interpretation, and the ATs will start confirming it instead of the contract: the acceptance level collapses into the verification level.

7. A full session end to end

To dispel the sense of "magic", here is an end-to-end scenario — what the agent does and where the human steps in. The order of steps is an illustration: the agent edition sets the preconditions of artifacts (its section 4), while the order of work is your organization's production model ([standard/01 §1.2.1](#)):

1. You hand the agent the TZ and **RENAR-AGENT-EN.md**. The agent registers the TZ.
2. A critic agent (a different model) runs an adversarial review → verdict "findings present".
3. The agent prepares an ADAPT draft with the findings and a draft ACTZ. The **Architect** brings the protocol into a form the client can understand, puts it forward, and signs it together with the client → the ACTZ is signed; the findings receive their **decided-in**; the **Architect** approves the ADAPT with their signature.
4. The agent derives **BR** → **SR** → **SPEC**, shows the relationship graph. The **Architect** approves QG-0.
5. A test agent writes the **TC** pairs; the pinning run is red; the tests are frozen.
6. A **different** agent writes the implementation until the tests turn green; the runner records the result (QG-2).
7. Before the trials an isolated agent derives the **AT**s from the effective TZ and runs them. All green → the product can be submitted.
8. The agent assembles the business outcome. The **stakeholder** accepts it (QG-4).

The human joins in to approve, not to type artifacts — but join in they must, and only they put a signature under an obligation.

8. Common mistakes

Mistake	Why it is bad	What to do instead
Asking the agent to "just write code" from the TZ	The source of truth is lost — code without requirements	Requirements first, then implementation derived from them
Skipping the adversarial review	"No ADAPT needed" becomes a silent assumption	The review is always mandatory; "no findings" is a recorded verdict of a different model
Letting the agent sign an ADAPT or an ACTZ	The agent is not the owner or a signatory	The signature is always a human: the Architect signs the ADAPT; the client and the contractor sign the ACTZ

Treating a finding as closed without a signed ACTZ	The interpretation rests on a decision the client never made	resolved only with a decided-in on a clause of a signed protocol
Sending the agent's raw output to the client as a protocol	The client signs what they do not understand	The Architect aggregates and rephrases the questions — in the language of obligations
Reconstructing SR from finished code	Source-of-truth inversion	Change the requirement → then the code; the exception is a justified bug-fix
Using one model to both generate and check	No independence of review	The critic is on a different model than the generator
One agent writes both the code and its test	The test checks the code, not the requirement	Different agents; the test is frozen before implementation and was red at least once
Letting the AT generator peek at the SRs or the ADAPT "for context"	The ATs start confirming the interpretation instead of the contract	Input: the effective TZ only; a different model; no internal loop whatsoever
Running ATs derived six months ago at the trials	The system is presented against a contract that no longer exists	ATs are regenerated before every round of trials

[← Back to the guide overview](#)

12. Migration to v1.1

Step 4 of the procedure of §13.9.3 for the v1.1 wave (ADR-015...ADR-026, GitLab #30–#54). For implementations that claimed RENAR v1.0 conformance. A minor release is an immediate trigger of re-assessment (§13.7.3): the v1.0 claim stands until re-assessed but does not carry over to v1.1 automatically.

Not to be confused with 10-migration-v1 — that is the historical v0 → v1.0 migration; this is the set of changes of one minor wave with identifiers and traceability preserved.

1. What changed — the map of the wave

#	Change	Where	Basis	Class for an implementation
1	The description set — the unit of approval, version and verification; BR / SR / SPEC / TC norm without <code>status</code> / <code>version</code> ; <code>set-version</code> N.M , linear history, version record	§10.5, §6.5.4, §6.6.4, §8.8, §9.9	ADR-024, #46, #48	breaking (schema)
2	QG-0 / QG-1 / QG-2 — objects: the set version / the TC implementation / the version on a product version; <code>automation.set-version</code> , <code>last-run.set-version</code>	§10.3, §9.3	ADR-024	breaking (schema)
3	§13.3.5 — pair coverage is checked at QG-0 of the set version (<code>coverage-presence</code>)	§13.3.5, §10.11.1	ADR-024, ADR-025	mandatory clause — re-assessment
4	Manifest: <code>set-version</code> , <code>confirmation: first-party</code> <code>second-party</code>	§13.4.2	ADR-024, ADR-017	breaking (manifest)
5	SPEC-UC — the twelfth type (<code>role: human</code> <code>agent</code> , <code>steps[].ref</code>); the step reference to a statement in SPEC-UI / SPEC-PROC; a <code>ux</code> TC per action through the interface	§8.3, §8.5.12, §9.8	ADR-018, #35, #34	closed list no. 7; new rules
6	MW — the manual walkthrough record (a class of evidence)	§9.20, reference/02 §8.3	ADR-018, #34, #50	additive
7	First-party confirmation — a concept as the input, ACTZ / AT have no subject; combining roles	§1.4.4, §5.5.5	ADR-017, #33	additive (a kind of applicability)
8	Reusable component — a system in its own tree; <code>assumptions[]</code> , <code>applies-to</code> , the	§6.14, §10.11.1	ADR-016, #30	additive

	uses[] edge			
9	implements[] — mandatory clause §13.3.8	§13.3.8	#30	mandatory clause — re-assessment
10	The screen list — the list in the SPEC-ARCH body, screens[] in SPEC-UI, the coverage rule; the definition of a screen	§8.5.1, §8.5.6.1, §10.7.2	ADR-023, #42, #45, #54	additive (structural completeness)
11	Coverage completeness of the SPEC mandatory body	§8.4.1	ADR-023, #42	additive
12	SPEC-ARCH and SPEC-SEC mandatory and non-empty at the system level	§8.3, §10.7.3	model 3.3	additive (structural completeness)
13	The controlled form of an SR statement — six forms (recommendation)	§6.6.3.1	ADR-022, ADR-026, #37	recommendation
14	The description language — chapter 15: operators, glossary discipline, prohibited constructions, atomicity; mandatory from RENAR-2	chapter 15, §11.5, §1.7.5 row 17	ADR-026, #51	mandatory from RENAR-2
15	automation.status — two values (automated / manual-pending with a deadline and a reason)	§9.3	#50	breaking (schema)
16	AR records the primary agent model (primary.*); ai-provenance composition per the canon	§4.10.1, §12.3	#38, #44	schema (ai-provenance)
17	Metric threshold calibration — by the data-sufficiency condition	§12.3	ADR-020	additive
18	The non-degeneracy meter of a new mandatory control	§13.9.4	ADR-021, #40	procedural
21	The statement address <id>#n : verifies[].statement in a TC (mandatory when the artifact has more than one statement), a step ref in address form, coverage-presence by addresses	§15.1.1, §9.3, §8.5.12.1	v1.1 review	additive (TC schema)
20	Norm and process: the standard governs artifacts, lifecycles and rules, not the order of work; agent editions — preconditions instead of steps	§1.2.1, §1.3 item 7, §7.6.1, §11.11	ADR-027, #55	editorial (the boundary of the norm)

2. What an implementation does — by class

2.1 Breaking schema changes (rows 1, 2, 4, 15)

1. **Remove status and version from BR / SR / SPEC / TC norms.** The state of an artifact is derived from the set version (§6.5.4). Migration script: delete the fields, fix the first set version 1.0 by a version record (§10.5.4) signed by the Architect, including every artifact that was approved +; artifacts in draft stay in the set's draft.
2. **TC:** `verifies[].version / requirement-version` → remove; add `automation.set-version` and `last-run.set-version` = the set version on which the implementation was created and run; `automation.status` → `automated` or `manual-pending` with `manual-pending-until / manual-pending-reason`. Check: `scripts/validate-schema-examples.js` (rules `verifies-legacy-version`, `invalid-tc-type`).
3. **Manifest:** add `set-version` (the current approved set version) and `confirmation` (`second-party` with a stakeholder, `first-party` with a concept as the input, §1.4.4); `renar-version: "1.1"`.
4. **Substrate hooks:** replace per-artifact state transitions with the set-version enforcement points (§10.11.1): `coverage-presence`, `implements` -edge and `uses` -edge validation; QG-1 — only the admission of a TC implementation.

2.2 Mandatory clauses (rows 3, 9) — re-assessment

The v1.0 claim MUST be re-assessed (§13.7.3): §13.3.5 is now checked on the set version as a whole (every statement of the version — a pair of TC norms in the same version), §13.3.8 requires `implements[]` for subsystem BRs. Self-assessment — [reference/08](#); the `mandatory-clauses-confirmed` keys are not renamed.

2.3 Additive changes (rows 5–8, 10–12, 16–18)

- **SPEC-UC:** scenarios that lived as a SPEC-UI user journey or a SPEC-PROC happy path and stitch several SRs / SPECS — move into SPEC-UC with `role` and a `ref` on every step; steps without `ref` violate structural completeness. SPEC-UI statements about an action through the interface — cover with `ux` TCs.
- **The screen list:** add a "Screen list" section to the `system` -level SPEC-ARCH from the description (not from code routes); `screens[]` in every SPEC-UI; an uncovered screen blocks version approval. A system without an interface records "no interface".
- **SPEC-ARCH / SPEC-SEC:** create if absent; the silence of the TZ about security is closed by a SPEC-SEC of your own, with client-observable consequences — through an ACTZ.
- **Component:** libraries and platforms described as subsystems or a "fourth level" — convert into a system in its own tree with `uses[]` at the consumers (§6.14); path B or C — by the structure of the organization.
- **First party:** implementations without a stakeholder (the former §1.5.4) with a written concept — the kind of applicability of §1.4.4, `confirmation: first-party`; without a concept — out of scope.
- **MW:** record manual scenario runs as `MW-NN`; no standing manual run of `ux` TCs is introduced.
- **ai-provenance / AR:** bring `generated-by` to the versioned format, add `primary.*` to the AR.

2.4 The description language (row 14) — mandatory from RENAR-2

For **RENAR-2** + claims: the normative sections of artifacts — per §15.2–§15.5. Order: (1) operators — replace synonyms ("shall", "needs to", "can") with the five words of the list; (2) terms — reduce to one form along the chain, divergences are **terminology** findings; (3) the constructions of §15.4 — reformulate; (4) atomicity — split statements from which exactly one pair of TCs cannot be written. The forms of §15.6 are a recommendation; approved versions are not reworded after the fact — the edits enter with the next minor set version.

Row 20 requires no action: if your production process differs from the scenarios in the guides, it conforms to the standard as long as the preconditions of artifacts hold (agent edition, section 4); artifacts and rules of your own are **declared-stricter**.

3. Order of work

Step	What	Check
A	Inventory: artifacts with <code>status</code> / <code>version</code> , TCs with <code>requirement-version</code> , SPEC-UIs without <code>screens[]</code> , SPEC-ARCHs without a list, missing SPEC-ARCH / SPEC-SEC	a frontmatter script
B	The first set version <code>1.0</code> : version record, the Architect's signature; field removal	<code>validate-schema-examples</code> , structural completeness §10.7.2
C	TC: <code>set-version</code> , <code>automation.status</code> from two values; QG-1 as implementation admission	a run, <code>last-run.set-version = 1.0</code>
D	The additive changes of §2.3 — as the next minor set version <code>1.1</code>	QG-0 of the version: <code>coverage-presence</code> , <code>screens</code> , <code>uses</code>
E	The description language (RENAR-2+) — as artifacts are edited	adversarial review, static check
F	Manifest <code>renar-version: "1.1"</code> , <code>set-version</code> , <code>confirmation</code> ; self-assessment reference/08; re-assessment of the claim	§13.7.3

4. Common pitfalls

- **Set version ≠ product version.** `set-version` is the version of the description; the product version lives in the verification record (§10.5.4).
- **A minor on every change.** The "large or small" rule is not accepted: every approved change is a minor; a major is a decision at a milestone with a full audit.
- **A list from code.** A screen list from implementation routes violates §2.3.1; a screen in the code without an entry is a drift and a backward finding.
- **Upper-case operators.** Prohibited in the Russian edition of artifacts (reference/06 §2.1 item 4); unambiguity comes from the closedness of the list.

5. Rollback

The v1.1 changes delete no data: the removed `status` / `version` fields are recoverable from the substrate history (V1); the set version record is additive. Returning to a v1.0 claim — roll back the manifest and repeat the self-assessment against v1.0.

6. Related documents

- ADR-024 (the batch model; §9 — the places of edits by task of the wave) and ADR-026 (the description language) — the standard's decision archive, available in the repository
 - [CHANGELOG](#) — the v1.1 entry
 - [reference/08](#) — conformance self-assessment
-

Guide RENAR 1.1 — [renar.tech](#)
