RENAR

Practical guides for adopting RENAR

RENAR · Practical Guide · Version 1.0-draft | 06.06.2026

Authors: Vadim Soglaev, Andrey Yumashev

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.*

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** §10.3 + **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 §27): 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
  adapt/                # ADAPT artifacts
  br/                   # Business Requirements
  sr/                   # System Requirements
  specs/{arch,api,data,ui,sec,ai,proc,int,ops}/
  tests/                # TC (tc-type incl. contract)
```

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 (Φ3-152).
```

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-ref: "<link>" }
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 — Ф3-152 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 Ф3-152.

### 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. ADAPT `approved` (5 min)

The client provides answers. The backward lifecycle: `open → asked-to-client → answered → resolved → frozen (after approval)`.

Summary after the answers:

```
### B-001: resolved (2026-05-18, J. Ivanov)
"Show a message + a resend button. A 401 with no explanation is bad UX."
→ a scenario added to Forward §2; SR-04 created.

### B-002: resolved (2026-05-18)
"Data center in the RF, RF jurisdiction. The provider is chosen by the
contractor."
→ data-residency: ["RU"] in Forward §2.

### B-003: resolved (2026-05-18)
"Resend no more than once every 5 minutes, no more than 5 times per day."
→ rate-limit rules in Forward §2; SR-04 refined.
```

All backward entries → `resolved`, `open-questions-count = 0`. **QG-ADAPT-approve** — 6 items passed:

```
status: approved
approval:
  client-signature: { signed-by: "J. Ivanov", role: PM, organization: "ClientCo",
signed-at: "2026-05-18T15:30:00Z", signature-ref: "<link>" }
  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.

Step 4. BR-01 (3 min)

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

```
---
id: BR-01
title: "User registration via email"
type: BR
status: approved
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 Ф3-152
compliance: [{ standard: "Ф3-152", 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
status: approved
parent: { id: BR-01 }
source: { adapt: "ADAPT-001", adapt-section: "Forward §2" }
constrained-by: ["SPEC-API-01", "SPEC-DATA-01", "SPEC-SEC-01"]
quality-characteristic: [functional-suitability, security]
---

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

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
status: ready
verifies: [{ id: SR-01, requirement-version: "1.0" }]
negative: false
automation: { status: automated, location:
"tests/auth/test_signup.py::test_signup_success", runner: pytest }
---

## Given
- 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
status: ready
verifies: [{ id: SR-01, requirement-version: "1.0" }]
negative: true
automation: { status: automated, location:
"tests/auth/test_signup.py::test_signup_invalid_email", runner: pytest }
---

## Given
- email "not-an-email" (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. Run the TCs and promote the SR (2 min)

The test runner on the substrate runs the TCs 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>", requirement-version: "1.0" }
```

When **all** TCs from `SR-01.verified-by[]` are green → spot-check (Core Rule 5: the engineer manually runs 1-2 random passing TCs and checks them against the SR). If the spot-check passes → **QG-2 Verification Gate** passed → SR-01 → `verified`:

```
# sr/SR-01-sign-up.md:
status: verified
verified-by: ["TC-01", "TC-02"]
verified-at: "2026-05-19T14:00:00Z"
verified-by-engineer: "P. Petrov"
```

Likewise — SR-02, SR-03, SR-04. When all SRs in BR-01 are verified → QG-4 (BR ready for acceptance).

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)
       └─ constrained-by SPEC-API-01 (REST contract)
                                  └─ depends-on SPEC-DATA-01, SPEC-SEC-01
```

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 → the client gave answers → approved.

- 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 two quality gates: QG-ADAPT-approve (≡ QG-3 Architecture) and QG-2 Verification Gate.

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 + 9 phases)	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 / Φ3-152 / AI Act	06-compliance.md
Failure modes — typical failures and patterns	07-failure-modes.md
The full normative specification (15 chapters)	`standard/`
Artifact schemas and validation rules	`reference/02-schemas.md`
Glossary and mapping to industry standards	`reference/01-glossary.md`

RENAR Quickstart 1.0-draft — renar.tech

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.*

Reader's route. Phases 0–2 — context gathering, signing the TZ, ADAPT. 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 the QG-4 acceptance loop.

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 (two-way 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 resolved or reclassified; the Forward covers §2 + §3 of the TZ in full.

2.4 ADAPT in status `approved`:

```yaml
---
id: ADAPT-001
title: "Adaptation of TZ-2026-042 — Login Flow AcmeCorp"
type: ADAPT
source-tz: { id: TZ-2026-042, signed-date: "2026-05-03", signed-by-client:
"AcmeCorp PM" }
status: approved
approval:
  client-signature: { signed-by: "A. A. Ivanova", role: "Product Lead",
organization: "AcmeCorp", signed-at: "2026-05-04T11:30:00Z" }
  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**.

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
status: approved
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 §14.1 Forward; 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 §14.1 Forward).

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

## Endpoints
```

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

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

### POST /auth/totp-setup — 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 (RENAR Core Rule 4: for each assertion in an SR — 1 pos + 1 neg TC).

TC-001 (positive):

```
---
id: TC-001
title: "Registration with an allowed domain — happy path"
type: TC
tc-type: system
status: ready
verifies: [{ id: SR-01, requirement-version: "1.0" }]
negative: false
automation: { status: automated, location:
"tests/auth/test_registration.py::test_allowed_domain_succeeds", runner: pytest }
---

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

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

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

```
## Not in 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
status: ready
verifies: [{ id: SR-01, requirement-version: "1.0" }]
negative: true
automation: { status: automated, location:
"tests/auth/test_registration.py::test_disallowed_domain_rejected", runner:
pytest }
---

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

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

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

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

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 BR-01 (`draft → approved`). Preconditions: `source.adapt = ADAPT-001 (approved)`; the SR tree in an admissible state before the BR is approved; the adversarial review succeeded; the assertions and links cite sections of ADAPT-001. Postcondition: BR-01 + child SRs, where needed, cascade `draft → approved`.

5.2 QG-1 — TC only: `draft → ready`. Per §10.3.2, QG-1 applies only to TC and separates a prepared test case with an executable verification implementation from a draft. Preconditions for a TC: the implementation version-pin is fixed (V5); `automation.status` + `location` are valid; static checks pass; pos/neg pairing (§9.7); the mandatory body sections are filled in. Postcondition: `draft → ready`.

After **QG-0** on BR/SR and **QG-1** on each TC, TR work can be opened (Phase 6).

Phase 6 — Implementation

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`, `requirement-version = 1.0` in the TC files.

7.2 Spot-check (Core Rule 5). Once per sprint the Engineer manually runs 5 random passing TCs and checks the actual result against the SR. Selected: TC-001, TC-008, TC-012, TC-019, TC-024 → 5/5 match.

7.3 Promote SR-01 → verified. QG-2 preconditions: approved ADAPT linkage; pos/neg TCs passing; `last-run.requirement-version` is fixed; the spot-check passed. Postcondition: SR-01 `approved → verified`; 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. 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). After 1 iteration with the client → approved.

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

```
Affected:
  BR-01 (scope extension)
  SR-01: verified → approved (TC rerun required)
  TC-001..004: re-pin 1.0 → 1.1; +2 new TC (subsidiary domain)
  TR-115: new implementation task
  SPEC-SEC-01: allowed-domains extended
```

8.4 Apply delta. 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. After the spot-check — SR-01 v1.1 → verified again.

> ***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`*. This removes the dual-signature overhead for trivial changes (e.g., a field rename).*

Phase 9 — QG-4 (acceptance)

9.1 Four weeks after release. Window: 4 weeks post-release.

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

9.2 QG-4 report + adversarial. The AI generates an acceptance report. The adversarial critic finds: "recovery via admin is not covered by a TC verifying the full flow with tickets." The Architect agrees → creates a mini-delta to add an acceptance TC.

9.3 Sign-off. After the findings are closed, the client signs the acceptance. BR → status `accepted`. Report archive: `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)
├── br/                     BR-01 + BR-02 (status: accepted)
├── sr/                     SR-01 (v1.1, verified) + SR-02..SR-07 (v1.0, verified)
├── specs/                  SPEC-UI-01 + SPEC-API-01 + SPEC-DATA-01 + SPEC-SEC-01
(verified; SPEC-SEC-01 v1.1)
├── tests/                  TC-001..TC-028 (28 TC, 100% passing)
├── tz/                     TZ-2026-042.md + TZ-2026-051.md (delta, immutable)
├── elicitation/            # Phase 0 artifacts
├── lessons/2026-Q2.md      # Phase 9 lessons
└── QG-4-REPORT-v1.0.md     # acceptance report
```

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%
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. **Closing the loop** — QG-4 ties the result to business metrics (KPI achievement).
6. **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-draft — 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 an ADAPT.

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. For each TZ an `ADAPT` is created — a bridge artifact: a Forward section ("how we understood it") and a backward section (questions for the client).
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/`, `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).

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**:

```yaml
---
id: BR-12
title: "Employee registration via corporate email"
status: approved          # 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. Lifecycle statuses are used for real: every artifact is in one of the closed states (`draft` / `approved` / `verified` / `deprecated` / `obsolete`).
3. TC are created for all priority=must BR / SR / SPEC.
4. A substrate-native COVERAGE report is auto-generated on every promote-transition.

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 approved artifacts have successfully passing test cases (`TC`); the transition 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 approved artifacts have a `verified-by` link to ≥ 1 TC.
2. Pos/neg pairing for every normative clause.
3. The `QG-2` gate (`Verification Gate`) is blocked by substrate-built-in checks: a transition to `verified` only when all `TC` s pass for the current `requirement-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 ≠ the implementation model.
7. `ai-provenance` for AI-generated artifacts: at minimum `generated-by` + `generated-at`.
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.

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 Core (5 rules + ADAPT) — on the order of half a day to a day per engineer.
- ADAPT discipline per TZ — additional hours on elicitation / backward 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: an ADAPT with a dual signature fixes the interpretation before code.
- 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/                ← REQUIREMENTS repository (a separate VCS repo)
│   ├── tz/                       ← TZ-YYYY-NNN.md (immutable after registration)
│   ├── adapt/                    ← ADAPT-NN.md (bridge artifacts)
│   ├── br/                       ← BR-NN.md
│   ├── sr/                       ← SR-NN.md
│   ├── specs/                    ← SPEC-* by subfolder (arch/, api/, data/,
int/, …)
│   │   ├── arch/  api/  data/  ui/  ai/  int/  proc/  sec/  ops/
│   ├── tr/                       ← TR-NN.md (task requirements)
│   ├── tests/                    ← TC-NN.md (test cases; `TC` are standalone
artifacts)
│   ├── 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 `main`; the `id:` 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	`git diff` + PR/MR review with a mandatory approve before merge
V4 — branching / change-set	A draft is kept separate from the approved truth	Feature branches (`draft` / `review`) against `main` (`approved`); a PR is a change-set
V5 — cross-substrate version pin	The implementation references a specific version of a requirement	Submodule pin between `.req` and `.src`; commit SHA / `requirement-version` in `verifies[]`
V6 — author & timestamp	Every change unit records its author and time	Commit metadata: author + date; for an AI agent — `ai-provenance`

> *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 current SR / SPEC through `verifies[].version`.
- New / changed TC match the contract in `.req`.
- No changes to TC pass/fail criteria without a `[test-spec-change]` tag.

Approval = QG-2 (standard/10 §10.3.3) — after the automated TC pass.

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 versions pinned to current requirement-version
  run: node scripts/validate-tc-version-pinning.js

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

7. 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. **ADAPT creation:** `adapt/ADAPT-001.md` — Forward (an interpretation per § of the TZ) + Backward (questions to the client), standard/07. Backward is worked through with the client before approval.
4. **Dual ADAPT signature → QG-ADAPT-approve:** ADAPT moves to `approved` (client + Architect), `open-questions-count == 0`.
5. **Decomposition:** the AI agent derives BR from the approved ADAPT, then SR (`source.adapt`), SPEC (`constrained-by[]`), and paired pos/neg TC.
6. **QG-0 approval** of each artifact (`draft → approved`); a CI dry-run of the new TC.
7. **PR into `.req` → merge;** the bot regenerates REQUIREMENTS.md / COVERAGE.md / TEST-PLAN.md.
8. **Set up `.src`:** add the submodule `requirements/ → <project>.req @ <commit>`, create TR, implement, QG-2.

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 + delta-ADAPT creation:** `tz/TZ-2026-042-delta.md` (the delta text) and `adapt/ADAPT-NNN-delta.md` (forward interpretation + backward findings, standard/07 §7.6). The delta-ADAPT MUST pass the dual signature before the affected requirements are modified.
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:** version++, status: `approved`, 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 TC — approval → the AI agent moves the requirement to `verified`.

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-draft:

- ✅ **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 + 9 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 ∈ {ru, 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 → ADAPT → BR) is not yet documented as a separate workflow alongside §7 (the delta workflow); step 2 in §7 explicitly uses the delta-ADAPT per standard/07 §7.6. The approved forward workflow is the subject of a separate draft chapter.
- ⏳ **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	`grep -r "^id: SR-12" sr/`
Find the TC verifying SR-12	`grep -rl "id: SR-12" tc/`
Find an orphan SR (no TC)	`node scripts/find-orphans.js sr`
Create a new SR from a template	`cp library/templates/sr.md sr/SR-NN.md && $EDITOR sr/SR-NN.md`
Diff frontmatter over a period	`git log --all --since=... -- sr/`
Current requirement-version of SR-12	`yq '.version' sr/SR-12.md`
List stale TC (last-run < current version)	`node scripts/list-stale-tc.js`

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
`[coverage]`	Post-merge in `.req`	Regeneration of COVERAGE.md / REQUIREMENTS.md / TEST-PLAN.md
`[baseline-update]`	After approval of a PR that changes a baseline	Regeneration of PNG baselines for SPEC-UI
`[bump-req]`	Submodule bump in `.src`	Only the submodule pointer + minimal metadata
`[reconcile]`	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 (§3.2–11.3).

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 `_id`
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 `author` + `updated_at` 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 (§3.5).

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 5 rules, ADAPT, the 2 QGs) 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	`<system>.req/br/` aggregated	Lean Portfolio Mgmt	All BR in the group with status `verified`
Capability	Subsystem BR (if the subsystem has its own stakeholder)	`<subsystem>.req/br/`	Solution Architect	All subsystem BR `verified`
Program Epic	Optionally — a large initiative inside the ART, aggregated SR	`<system>.req/sr/` aggregated	Product Manager	The target outcome metric achieved
Feature	SR (or a linked group of SR)	`<subsystem>.req/sr/`	Product Owner	TC from `verified-by` green on the current `requirement-version` (QG-2)
Story	TR (Task Requirement)	`<subsystem>.req/tr/` or task tracker (Jira, Linear, GitLab Issues)	Team	All AC met, evidence recorded, code merged
Enabler Epic	SPEC-AI / SPEC-ARCH / SPEC-OPS	`<system>.req/specs/`	Architect	Eval metrics within thresholds; baseline recorded
Spike	TR with `level: research` + a Decision in the decision log	`<subsystem>.req/tr/` + decision log	Team	Decision recorded and linked to the originating SR
Defect	TR + a reference to the violated SR via `defect-of`	task tracker + `linked_defects` in the SR	Team	Negative TC passes, regression test added

Forbidden mappings: INT-SR is a deprecated term (§4.3 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

```
WSJF = (User-Business Value + Time Criticality + Risk Reduction & Opportunity Enablement) / 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

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

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`.

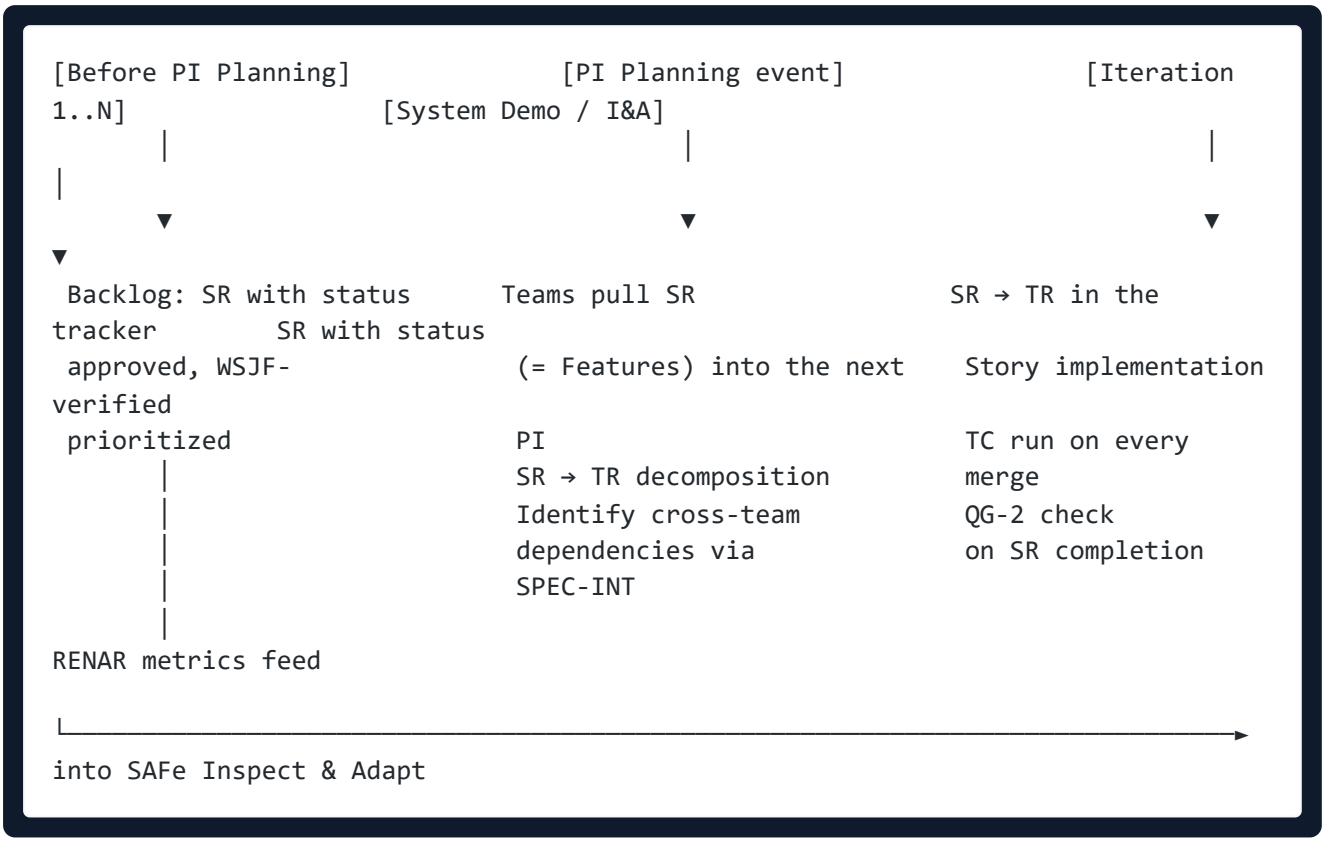
---

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 with status `proposed` or `approved`	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 `in-progress`
System Demo	SR with status `verified`	Evidence from 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 (readiness to start) and QG-2 (verified)
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

```yaml
# In <subsystem-a>.req/sr/SR-05.md
---
id: SR-05
parent: BR-12
title: "User registration via corporate email"
status: approved
constrained-by:
  - id: SPEC-INT-01
    ref: "specs/int/SPEC-INT-01-auth-handshake.md"
    requirement-version: "1.2"
verified-by:
  - TC-23
```

```
  - TC-24
---
```

6.2 SPEC-INT 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"
status: approved
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 in the group → `verified`, KPI impact confirmed via an outcome metric
Capability	Subsystem BR	All BR with status `verified`; outcome metric measured
Feature	SR	QG-2 passed: every TC from `verified-by` has `last-run.result = pass` on the current `requirement-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 are created **before** implementation; QG-0 requires `verified-by` to be empty only when `status: proposed`, not `approved`
Refactoring	The continuous reconciliation hook (§7.5 ADAPT); 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 (SR 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"	**Drift Lag** (reconciliation)

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 9 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 implemented as a mandatory stage before decomposition into BR.
- **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 + 9 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 9 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`; `requirement-version`. 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 `compliance: A.5.7`
A.5.8 Information security in project management	RENAR itself as process compliance
A.5.34 Privacy and protection of PII	SR with encryption + `data-classification.contains-pii: true`
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 + `retention-days`
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 `gdpr.lawful-basis: consent / contract / legal-obligation / vital-interests / public-task / legitimate-interests`
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

GPAI 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 GPAI 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 `business-context`, `data-classification`, `ai-act.risk-class`
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 `eval-tests`
§6.4.4 Risk treatment	SR-mitigations + post-mitigation evaluation
§6.4.5 Communication and consultation	RACI matrix (05-safe-comparison §9)
§6.4.6 Monitoring and review	Reconciliation hook (drift detection)

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 + ADAPT + BR
Architecture definition	SPEC-ARCH + SPEC-AI
Design definition	SPEC-AI (model card, prompt design, RAG)
Implementation	TR + implementation
Verification	TC (`tc-type: eval` for AI)
Validation	Acceptance TC + spot-check
Operation	Reconciliation hook (drift detection, §4.11)
Maintenance	Delta-TZ + 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	`contains-financial: true` + 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 `tc-type: security` + pen-test (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 exactly one approved ADAPT with dual signature.
- All 9 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 §14).
- `assessment-date` is recent, `next-assessment-due` is not overdue.

All 7 are `yes` → the project is conformant to the declared `level`. At least one `no` → the manifest is not issued (§13.5.2).

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

```
renar-version: "1.0"
senar-version: "1.0"
manifest-version: 1
manifest-id: "CFM-2026-014"
level: "RENAR-2"
level-target: "RENAR-3"
assessment-mode: "self"
assessment-date: "2026-05-20"
assessor: { id: "architect-team-lead", role: "architect", signature-ref: "
<pointer>" }
next-assessment-due: "2026-08-20"
mandatory-clauses-confirmed:
  sot-inversion: true
  substrate-v1-v6: { v1: true, v2: true, v3: true, v4: true, v5: true, v6: true }
  adapt-per-tz: true
  spec-types-closed-list: true
  tc-pos-neg-pairing: true
  quality-gates-closed-list: true
  closed-lists-backward-findings: true
quality-gates: { qg-0: required, qg-1: required, qg-2: required, qg-3: absent,
qg-4: absent }
spec-types-supported: ["SPEC-ARCH","SPEC-API","SPEC-DATA","SPEC-INT","SPEC-
PROC","SPEC-UI","SPEC-AI","SPEC-SEC","SPEC-OPS"]
exceptions: []
replaced-by: null
```

11. Auto-generated compliance artifacts

Artifacts generated from existing requirements for auditors:

Artifact	Source	Contents
Records of Processing (GDPR Art.30)	BR/SR with `contains-pii: true`	Data categories, lawful basis, retention, recipients
Statement of Applicability (ISO 27001)	BR/SR with `compliance: ISO 27001:2022`	Annex A controls in/out of scope, justification
Data Inventory	All `data-classification` 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 `dpia/` 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 §14.

---

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-residency.region: eu-west`), 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: Statuses (`proposed` / `approved` / `verified` / `obsolete`) 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 exist and are `verified` .

Prevention: Requirement IDs are **immutable** — renaming is forbidden. Deprecated requirements stay in the repository with status `obsolete` ; 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 requirement has already changed — `last-run.requirement-version` is lower than the requirement's current `version`. The test is green, but it checks outdated behavior.

Detection: The coverage report shows a "Stale" category — TCs with an outdated `last-run.requirement-version`. Reconciliation catches this automatically (via the V5 version pin).

Prevention: A TC has the mandatory field `verifies[].requirement-version` — a pinned version. QG-2 forbids moving a requirement to `verified` if at least one TC in `verified-by` has a stale `last-run`.

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 `ai-provenance.model`; no multi-model agreement
AIR-06	Test-fitting / greening tests	High	diff in TC pass/fail without a `[test-spec-change]` 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 tool_event 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 `approved` TC in scope has a `tc-type` and `verified-by[]`
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 §5.6)
4. Re-run	AI agent or human	Updated TCs + diff	QG-2 pre-condition: `passing-tests / total-tests` for the scope (§9.10)
5. Audit trail	substrate (V1)	A commit/change unit with the `adversarial-review` 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 — §4.5 (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 ADAPT as a formality

Symptom: The client / Stakeholder does not read the ADAPT before signing. The backward section (questions for the client) is empty or contains yes/no answers without context.

Sign: An ADAPT approved < 24 hours after generation; the rate of disputed requirements at acceptance is rising.

Mitigation: Dual signature on the ADAPT (standard/05-roles §5.5) — both the Stakeholder and the Architect are required. The backward section MUST contain ≥ 1 non-rhetorical question. Spot-check 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). A TC without a `last-run` for N days is automatically marked `stale`; QG-2 blocks until they are re-run.

---

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-run the TC against the current requirement-version.
- Test-fitting → revert the criteria; root-cause the AI agent.
- Organizational → process retro + the specific mitigations (§5).

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 §2 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 (§5).** 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 §5.7.** 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.

---

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	`.req`	BR, SR — what the system must do	Architect, Tech Lead
Source code	`.src`	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 §14: 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; add an entry to
source[]
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 §14.4; 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 — the 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	`feat/TASK-NN-slug`
Bug fix	`fix/TASK-NN-slug`
Changing a requirement	`change/TZ-YYYY-NNN-slug`
New requirement	`feat/BR-NN-slug` or `feat/SR-NN-slug`

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) → create the `feat/TASK-NN-slug` branch in `.src` → implement + write tests for the AC → MR/PR → review → merge.

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-index.md` (the list of changed requirements) → 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
`acme-platform.req` (system BR/SR)	Read-only	Read-only	Write via MR
`acme-portal.req` (subsystem SR)	Read-only	Write via MR	Write via MR
`acme-portal.src` (code)	Write via MR	Write + review	—
`acme-ai.req` (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?	`acme-portal.req/sr/SR-NN-*.md`
Why does the business need it?	`acme-platform.req/br/BR-NN-*.md`
Where did this requirement come from?	frontmatter `source` → TZ
How did the requirement change?	`git log -- sr/SR-NN-*.md`
How does the integration with Acme AI work?	`acme-platform.req/specs/int/SPEC-INT-01-*.md`
What changed with the latest TZ?	`acme-platform.req/tz/TZ-YYYY-NNN-index.md`

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

> *Six scenarios (E1–E6): three full in-doc cycles (E3–E6) plus two walkthroughs (E1–E2). Each in-doc example: **TZ → ADAPT → BR/SR → SPEC → TC → QG**.*

#	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)	§14	Legal, PM, Architect	Compliance, SPEC-DATA/SEC
E4	Webhook idempotency (REST API)	§4	Backend, Architect	`tc-type: contract`, SPEC-API
E5	RAG assistant (SPEC-AI eval)	§5	AI engineer	`tc-type: eval`, judge isolation
E6	Delta-TZ: scope dispute	§5	PM, Client, Architect	ADAPT backward, immutable TZ

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)

```yaml
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 `privacy-officer` + an audit log

2.3 BR + SR

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

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

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

2.4 SPEC (excerpts)

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

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
status: ready
verifies:
  - id: SR-08
    requirement-version: "1.0"
negative: false
---
## When
POST /v1/privacy/export (session: verified-user)
## Then
- 202 + job_id; status pending; audit event recorded
```

```
---
id: TC-081
title: "Export rejected without an MFA-verified session"
type: TC
tc-type: security
status: ready
verifies:
  - id: SR-08
    requirement-version: "1.0"
negative: true
---
## When
POST /v1/privacy/export (session: password-only, no MFA)
## Then
- 403; job 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 `approved`, backward findings closed
QG-2 Verification Gate	`TC-080` … `081` pass; the `requirement-version` in `last-run` matches (`1.0`)

2.7 What next

- The full scenario under `git` — 03-tool-guide-git
- Compliance mapping — 06-compliance
- Conformance manifest — reference/08

---

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:

```yaml
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):

```yaml
id: TC-200
type: TC
tc-type: contract
verifies: [{ id: SR-20, requirement-version: "1.0" }]
negative: false
```

```yaml
id: TC-201
type: TC
tc-type: contract
verifies: [{ id: SR-20, requirement-version: "1.0" }]
negative: true
```

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

Quality gate `QG-2`: both `TC`s pass; the `requirement-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):

```yaml
id: TC-300
type: TC
tc-type: eval
verifies: [{ id: SPEC-AI-02, requirement-version: "1.0" }]
automation:
  judge-model: "eval-judge-v2"   # ≠ production chat model
negative: false
```

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

See standard/09 §9.6.2, guide/07 §4.5.

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** → a new ADAPT-002b (forward + backward).
3. Backward finding: "PDF export was not part of Forward §3 of ADAPT-002."
4. The client signs the delta-ADAPT → new SR/SPEC/TC.

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)

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

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

source:
  adapt: ADAPT-NOTIFY-001
  adapt-section: "Forward §2"
  tz-section: "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-NOTIFY-API-01, SPEC-NOTIFY-OPS-01
```

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 are in status `approved` (or `verified`).
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.

---

10. Migration to v1.0-draft

> *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
`verifies[].version` without `requirement-version`	Update the TC frontmatter (reference/02 §8)
Manifest `renar-version: 0.1-draft` or absent	Issue a new manifest (reference/08)

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

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 `specs/ui/baselines/`
`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

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 §14 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: `source.adapt` on the approved ADAPT.

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 §14.

Phase D — manifest (1 day)

1. Bump `renar-version: "1.0-draft"`.
2. Increment `manifest-version`; new `manifest-id`.
3. Signature of the Architect / Tech Lead (V6).

---

4. Common pitfalls

Mistake	Consequence	Fix
Renaming `SR-05` → `SR-05-v2`	V1 immutable-ID violation	`deprecated` + a new ID with `replaces`
Leaving `type: system` on a TC	Validator fail; KG drift	`type: TC + tc-type: system`
Migrating code before the `.req`	SoT inversion violated	First the SR/SPEC/TC approved, then the TR
Skipping ADAPT "only for new TZs"	Non-conformant for legacy TZ	A retrospective ADAPT for every active TZ

---

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-draft — renar.tech