The *AI-Native* SDLC Framework

A 7-phase lifecycle that ensures cross-artefact system integrity throughout AI-augmented development — model-agnostic, grounded in four real drift incidents, and designed to complement enterprise delivery frameworks.

↓ PDF ↓ DOCX

INTERACTIVE · CLICK EACH PHASE

The 7-Phase *Lifecycle*

Each phase has a trigger, an AI role, artefacts produced, and an exit gate. Click a phase tab to explore it in detail.

01	02	03	04	05	06	07
FRAME	**SEAL**	**SPEC**	**BUILD**	**VERIFY**	**RECONCILE**	**EVOLVE**

PHASE 01 · FRAME

FRAME

Trigger: A new product, feature, or module is requested. No code is written yet.

AI Role: Socratic partner — ask questions, reflect back assumptions, surface contradictions. Do not produce solutions.

EXIT GATE

You can state the Problem, Core Bet, and Success Metric out loud in under 90 seconds without referring to any document.

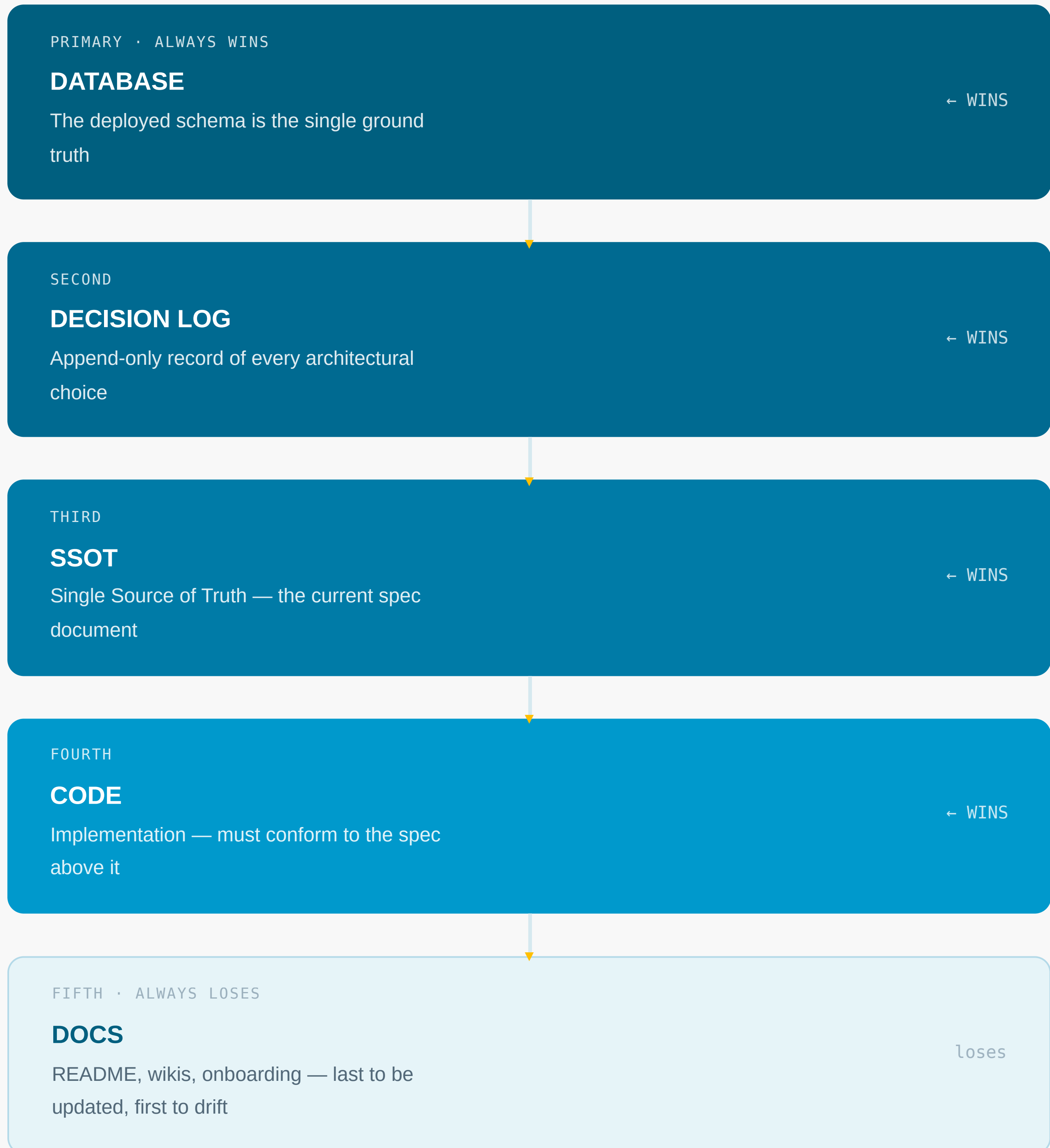
ARTEFACTS PRODUCED

- → Problem Statement (1 paragraph, plain language)
- → User & Context brief — who, what device, what constraints
- → Core Bet — the testable thesis this work assumes
- → Non-Goals — explicit list of what this is NOT
- → Success Metric — a single measurable number

CONFLICT RESOLUTION

The *Authority Chain*

When two artefacts disagree, the left always wins. Declared once, enforced permanently.



DESIGN PRINCIPLES

Seven rules that govern every *decision.*

P1 Model-Agnostic by Default

No artefact, template, or gate depends on a specific AI model. Models are interchangeable tools; the process is the constant.

P2 Artefacts Over Conversations

Every decision, contract, and assumption lives in a versioned file. Chat history is a side-effect, not a source of truth.

P3 Authority Chain is Explicit

DATABASE > DECISION LOG > SSOT > CODE > DOCS. When artefacts conflict, the left wins. Declared and enforced, not inferred.

P4 Sealed Interfaces, Mutable Implementations

Lock the contract, not the code. New capabilities extend sealed interfaces; they do not modify them.

P5 Verification is Non-Negotiable

Every AI-produced artefact passes through at least one verification gate before entering production or the Decision Log.

P6 One Unit of Work, One Chat

One chapter, one bug class, one module — one isolated chat. Long chats drift. Short focused chats don't.

P7

Rollback is First-Class

Canonical backups, sealed baselines, and named archives exist at every milestone. Undo is a process capability, not a developer superpower.

ARCHITECTURAL ENFORCEMENT

SEAL v3 — Five Rings of *Physical Enforcement.*

Process rules are enforced by humans. SEAL rings are enforced by the system. The distinction matters — deadline pressure degrades human discipline. It does not degrade a git hook.

R1

Self-Describing File Headers

Human-readable · Zero tooling required

Prevents: FM-07
Accidental edit

Every file in the `sealed/` directory carries a header declaring its sealed status, version, and the authority that sealed it. An AI reading the file encounters the contract declaration before any implementation — making accidental mutation visible even without tooling.

```
// SEALED CONTRACT — DO NOT MODIFY
// Authority: SSOT v4.7 §E2 · Sealed: 2026-01-14
// Hash: sha256:a3f8c2... · Ring: 1/5
// Modification requires: major version bump + arch review
```

R2

Physical Directory Isolation

Filesystem-level · Visible in IDE + Git

Prevents: CD1
Count drift

Sealed files live in a dedicated `sealed/` directory, physically separated from mutable code. This separation is visible in every IDE, every diff, every PR. A change to `sealed/` is immediately obvious — not hidden in a large changeset. New capabilities are added in `mutable/` via extension pattern, never by editing sealed files.

```
app/src/main/java/com/miva/
├── data/
│ ├── sealed/ # IMMUTABLE — contracts only
```

```
│ │ ├── SealedApi.kt
│ │ ├── contracts/
│ │ └── model/
│ └── mutable/ # all implementation goes here
│ ├── api/
│ └── network/
```

R3

Git Pre-Commit Guard

Automated · Blocks commit if sealed/ touched

Prevents: FM-10 Silent mutation

A git pre-commit hook (`seal-guard.sh`) runs before every commit. It inspects the staged diff for any changes to files under `sealed/` . If any are found, the commit is rejected with a clear error message citing the specific file and the required authority approval path. The guard cannot be bypassed without `--no-verify` , which is logged and flagged in the next reconciliation sweep.

```
✗ SEAL GUARD: Commit rejected
Modified sealed file detected:
sealed/contracts/SealedApi.kt
Sealed files require:
1. Major version bump in SSOT
2. Architecture review board approval
3. Decision Log entry before commit
Use --no-verify ONLY with arch approval logged.
```

R4

Build-Time Hash Verification

CI/CD level · Fails build if hash mismatch

Prevents: CD1 FM-10

At seal time, a SHA-256 hash is computed for each sealed file and stored in `SealedIntegrityCheck.kt` . Every build re-computes the hash and compares. Any mismatch — even a single changed character — fails the build immediately. This catches changes that bypassed Ring 3 (e.g. via `--no-verify`) and changes made directly to files without going through git.

```
// SealedIntegrityCheck.kt
val SEALED_HASHES = mapOf(
```

```
  "SealedApi.kt" to "a3f8c2d...",
  "Models.kt" to "b9e21a4...",
  "GrantContracts.kt" to "c7f3b8e..."
)

// Verified at app startup + build time
fun verifyIntegrity(): SealStatus
```

R5

Runtime Integrity Verification

Production · Detects tampering at runtime

Final catch
All rings

At application startup, `SealedContractVerifier.kt` re-runs the hash check against the embedded baseline. Any mismatch prevents the app from starting in production mode. In debug mode, it logs a warning with the specific file and delta. This ring is the final catch — it operates entirely at runtime, independent of the build system, catching anything that slipped through rings 1–4.

```
// Runs at app startup — before any feature
fun verifyAll(): SealStatus {
  return SEALED_HASHES.entries.map { (file,
expected) ->
    val actual = sha256(loadFile(file))
    if (actual != expected) SealBreach(file,
actual, expected)
    else SealIntact(file)
  }.let { results ->
    if (results.any { it is SealBreach })
INTEGRITY_FAILED
    else INTEGRITY_CONFIRMED
  }
}
```

THE EXTENSION PATTERN

Eternal interfaces. *Infinite evolution.*

Sealed contracts are eternal — never deprecated, never removed. When new capability is needed, it is added via an extension that calls the sealed contract internally. The original contract remains alive,

backward-compatible, and unchanged. This is how MiVA survived five model generations without a single sealed interface breaking.

✗ **FORBIDDEN — MUTATION**

```
// Changing the sealed function
// breaks every caller
fun getUser(id: String): User
fun getUser(id: String,
            withPrefs: Boolean):
User
```

✓ **CORRECT — EXTENSION**

```
// Sealed function untouched
fun getUser(id: String): User

// New capability as extension
fun getUserWithPrefs(id: String) =
  getUser(id).also { loadPrefs(it)
}
```

WHY ARCHITECTURE BEATS PROCESS ALONE

What each layer *actually catches.*

DRIFT SCENARIO	PROCESS LAYER CATCHES?	SEAL ARCHITECTURE CATCHES?	HOW
AI edits sealed file accidentally	**No — needs manual review**	**Yes — Ring 3 blocks commit**	Git pre-commit hook
Sealed function count mismatch (CD1)	**Only if reconciliation runs**	**Yes — Ring 4 fails build**	Hash verification
Contract changed via --no-verify	**No**	**Yes — Ring 4 + 5**	Build + runtime hash

DRIFT SCENARIO	PROCESS LAYER CATCHES?	SEAL ARCHITECTURE CATCHES?	HOW
Spec says 19 functions, code has 23	**Only in Reconcile phase**	**Yes — SealedContractTest**	Test fails at build
New model changes AI behaviour	**Yes — model transition ritual**	**Yes — artefacts unchanged**	Both layers
Spec and code diverge over weeks	**Yes — weekly sweep**	**Partial — sealed only**	Process layer essential here

THE CORE MECHANISM

Continuous Reconciliation is not a phase. It's a *system.*

RECONCILE appears as Phase 6 in the lifecycle. But it is actually the mechanism that makes the entire framework work. Drift doesn't accumulate between phases — it accumulates continuously. The reconciliation system responds to that reality.

Three trigger types — *continuous coverage.*

Fires on specific development events. Checks only the artefact pair directly affected by that event. Fast, low-overhead, catches drift at the point of introduction.

- `git commit → Pair D (sealed hash)`
- `PR opened → Pair A (code vs SSOT)`
- `Schema migration → Pair C (DB vs code)`
- `AI session closed → Decision Log check`

Runs all five reconciliation pairs on a fixed cadence regardless of events. Catches drift that accumulated between event triggers — the slow drift that no single commit reveals.

- `Daily: Pair A + Pair D (15 min)`
- `Weekly: All 5 pairs (90 min)`
- `Per milestone: All pairs + zero-delta gate`
- `Per model release: All pairs + Pair E`

Threshold Alert

Automatic · Escalating

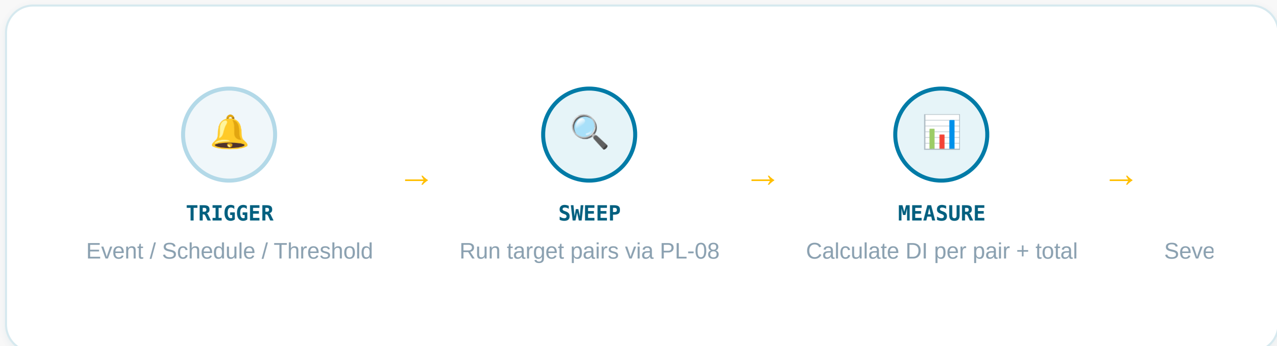
Fires when drift metrics cross defined thresholds. Escalates severity automatically. Blocks milestone gates if critical thresholds are breached.

- `DI > 0.1 → Weekly sweep triggered`
- `DI > 0.3 → Drift Audit recommended`

```
DI > 0.6 → Milestone gate blocked
```

```
Sealed breach → Build fails
immediately
```

The reconciliation *system flow.*



Automation *integration hooks.*

GIT HOOKS (RINGS 3 + 4)

```
# .git/hooks/pre-commit
#!/bin/bash
./scripts/seal-guard.sh # Ring 3
./scripts/hash-verify.sh # Ring 4
# Exit 1 = blocks commit

# .git/hooks/pre-push
#!/bin/bash
./scripts/pair-a-quick.sh # Code vs
SSOT
```

CI/CD PIPELINE STEP

```
# GitHub Actions / GitLab CI
- name: Drift Reconciliation
  run: |
    ./scripts/drift-sweep.sh --pairs
A,C,D
    ./scripts/di-calculate.sh
    if [ $DI_SCORE > 0.6 ]; then
      exit 1 # Block pipeline
    fi
```

Current status in MiVA: Rings 3 and 4 are implemented (git pre-commit guard + build-time hash). Weekly PL-08 sweeps are manual. CI pipeline integration for Pairs A and C is on the roadmap — the scripts exist, the automation step is the next increment.

ROLES & RESPONSIBILITIES

RACI — AI is a *role*, not a tool.

AI-Builder and AI-Reviewer are different chat sessions, ideally from different model families.

PHASE	HUMAN OWNER	AI BUILDER	AI REVIEWER	EXTERNAL
FRAME	R/A	C	–	I
SEAL	R/A	C	C	C
SPEC	A	R	C	I
BUILD	A	R	–	–
VERIFY	A	–	R	C
RECONCILE	A	–	R	I
EVOLVE	R/A	C	C	I

R = Responsible · A = Accountable · C = Consulted · I = Informed

WHAT TO MEASURE

Metrics that *actually* matter.

✓ Track these

Metric	Target
Drift incidents per sprint Leading indicator of process decay	≤ 0
Ready → merge time Long queues = verification too heavy	< 24h
BUILD chats under 60% context Near-limit chats spike hallucination	> 80%
Sealed contracts violated Any violation = SEV1	0%
Decision Log entries / week Too few = ad-hoc; too many = unsealed	3–10

✗ Never track these

Lines of code per day — AI makes this meaningless and harmful to optimise

Number of AI interactions — rewards chattiness over clarity

Speed to first commit — rewards skipping FRAME and SEAL

Token spend in isolation — a cheap hallucinated output costs more than an expensive verified one

· MiVA Education

Released under CC BY 4.0