

# Coherence-Driven Development

A role discipline for shipping software with AI agents.  
Not a framework. A way of working that improves itself.

cnos · 2026

# How agent coding **works today**

You give an agent a task. It writes code. You eyeball the diff. You merge.  
Repeat.

# The typical agent workflow

```
# The loop most teams run
1. Open issue or describe task
2. Agent writes code
3. Human reviews diff (or doesn't)
4. Merge to main
5. Hope CI catches problems
6. Repeat
# When it breaks:
"The agent said it was done"
"I thought CI would catch it"
"We'll fix it next time"
```

## What's implicit

- The agent is implementer, reviewer, and coordinator — all at once
- Quality depends on how carefully the human reads the diff
- "Review" means "skim and approve" at scale
- No separation between doing the work and judging the work
- Process improvements live in human memory, not in the system
- Each session starts fresh with no accumulated discipline

# What goes wrong — real examples

## Self-review is no review

- ✗ Agent implements a feature and declares "all tests pass"
- ✗ But the tests were written by the same agent in the same context
- ✗ They test what was built, not what was specified
- ✗ Confirmation bias, but at machine speed

## CI is not a quality gate

- ✗ Code merges, CI runs, CI fails — nobody notices
- ✗ 6 consecutive red builds before a human checks
- ✗ "CI will catch it" means "nobody is responsible"

## No institutional memory

- ✗ Same class of bug ships three times
- ✗ "We fixed this before" — but where? How?
- ✗ Learnings live in chat history, not in the process
- ✗ New sessions repeat old mistakes

## Coordination by heroics

- ✗ The human operator holds everything in their head
- ✗ Issue selection, sequencing, recovery — all manual
- ✗ Throughput limited by operator attention, not agent capacity
- ✗ Take a day off, the system stops

# The core insight

The problem isn't that agents write bad code.

The problem is that **nobody** is structurally responsible for catching when they do.

## What's missing

- **Separation of concerns:** the thing that does the work should not judge the work
- **Evidence over assertion:** "it's done" should mean something testable
- **Durable memory:** what we learn should change how we work, not just what we remember
- **Mechanical coordination:** sequencing work shouldn't require a human in the loop at every step

**CDD's answer:** define roles with structural responsibilities. Each role has a clear domain, clear artifacts, and clear constraints. The roles form a ladder — each level observes and improves the one below it.

This isn't new for humans. Code review, project management, retrospectives — these exist for the same reasons. CDD makes them **structural for agents**: loaded from files at dispatch time, not recalled from training.

# Five roles, one ladder

Each role's frame is the previous role's content. No role at level N can self-assess at level N+1.

- $\alpha$**  implements the deliverable — code, prose, config, design
- $\beta$**  reviews what  $\alpha$  produced — against acceptance criteria and contract
- $\gamma$**  coordinates the  $\alpha \leftrightarrow \beta$  loop within one cycle — scaffolds, dispatches, closes out
- $\delta$**  operates the sequence of  $\gamma$ -loops across cycles — selects work, holds gates
- $\varepsilon$**  iterates the  $\delta$ -discipline itself — surfaces protocol gaps, codifies improvements

**Roles are hats, not entities.** The same agent or person can hold different roles in different cycles. Within a single cycle,  $\alpha$  and  $\beta$  must be distinct — the independence is the mechanism by which review adds information.

# $\alpha + \beta$ — the minimum viable cycle

## $\alpha$ — the implementer

- Works on a cycle branch, never main
- Reads the issue, loads skill files, implements
- Writes `self-coherence.md` — traces gap → plan → evidence
- Honest-claim rule: grep before asserting something exists or doesn't
- Commits early, commits often
- Signals review-readiness when done

**$\alpha$  does not review its own work.** Self-assessment at the implementation level is structurally unreliable. That's  $\beta$ 's job.

## $\beta$ — the reviewer

- Reviews against acceptance criteria, not taste
- Contract integrity → AC walk → diff inspect → CI check
- Emits **APPROVED** or **REQUEST CHANGES**
- Findings cite evidence — line numbers, not vibes
- CI must be green before APPROVED (not after merge)

## Fix rounds

- REQUEST CHANGES →  $\alpha$  fixes →  $\beta$  re-reviews
- Max 3 rounds (if still failing, escalate)
- Each round is focused: fix the findings, nothing else

# $\gamma$ — the coordinator

$\gamma$  owns the  $\alpha \leftrightarrow \beta$  loop. The operator talks to  $\gamma$ ;  $\alpha$  and  $\beta$  are encapsulated behind it.



## What $\gamma$ does

- Creates cycle branch and scaffold artifacts
- Verifies the gap claim before starting (peer-enumeration)
- Dispatches  $\alpha$ , waits, dispatches  $\beta$
- Handles fix rounds if  $\beta$  returns REQUEST CHANGES
- Merges on APPROVED, writes close-outs
- Runs protocol iteration ( $\epsilon$  work) — captures findings

## Why $\gamma$ exists

- Without  $\gamma$ : the operator manually sequences  $\alpha \rightarrow \beta \rightarrow \text{fix} \rightarrow \beta$
- With  $\gamma$ : the operator says "work issue #352" and gets a result
- $\gamma$  absorbs the coordination overhead that would otherwise be ad-hoc
- Silence rule:  $\gamma$  surfaces only decision-points, not every step

**Add  $\gamma$  when** you're running 2+ cycles/day and the  $\alpha \leftrightarrow \beta$  dispatch overhead is eating your time.



# $\delta + \epsilon$ — operations and self-improvement

## $\delta$ — the operator

- Selects which issues to work and in what order
- Plans waves (multi-cycle batches of related issues)
- Dispatches  $\gamma$  — once per issue, or as an autonomous wave
- Holds external gates: push to main, tag, release, deploy
- Recovers from failures (timeouts, SIGTERM, merge conflicts)

**$\delta$ 's algorithm is mechanical.** Select issue → dispatch  $\gamma$  → wait → merge → next. Mechanical enough to automate. The first autonomous wave ran 5 issues in 2.5 hours.

## $\epsilon$ — the protocol iterator

- Observes whether CDD itself is working
- Writes `cdd-iteration.md` after each cycle
- Classifies findings: skill gap, protocol gap, tooling gap
- MCA discipline: ship a fix now, file an issue, or drop explicitly

## The self-improvement loop

```
run wave → observe failures → capture in iteration file  
→ file as issues with ACs → run next wave on those  
issues → codify into skill files → future waves load  
codified skills → repeat
```

$\epsilon$  often collapses onto  $\delta$ . Separate when iteration volume demands it.

# Skills, not memory

Each role reads its **SKILL.md** at dispatch time.  
Memory of the skill is not the same as having it loaded.  
The file is the constraint. It changes between sessions.

## What a skill file contains

- The role's exact algorithm (steps, phases, gates)
- Constraints and invariants (what's prohibited)
- Artifact contracts (what must be produced)
- Decision-point rules (when to escalate)
- Grading rubrics (how the role is assessed)

## Why this matters

- **Agents wake up fresh.** Without skill files, each session re-derives the process from scratch.
- **Skills are versionable.** When  $\epsilon$  patches a skill, every future cycle runs the patched version.
- **Skills are auditable.** You can diff what  $\alpha$  was told to do vs. what it did.
- **Skills are portable.** Different agents, same discipline.

This is how the self-improvement loop has teeth.  $\epsilon$  doesn't just "learn" — it **edits the file that  $\alpha$  loads next time.**

# CDD in practice — one day, two waves

May 12, 2026. One operator. Real output.

## Wave 1: CDD Tooling

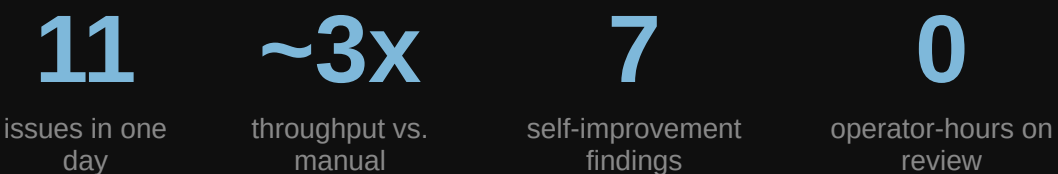
```
| # | Issue | Rounds | |-----|-----|-----|
----| | 286 | Encapsulate α/β behind γ | 2 | | 295 | cn
dispatch primitive | 1 | | 294 | cn cdd status N | 1 | | 293
| cycle-tag disambiguation | 1 | | 296 | issue-edit cache-
bust | 2 | | 305 | extend cdd-verify | 1 | 6 issues. 8
rounds. All merged. δ ran as autonomous agent — 2.5 hours.
```

## What wave 1 surfaced

- CI was red for 6 commits — nobody noticed
- Manual edits during wave corrupted working tree
- Wave coordination lived in /tmp, not the repo

## Wave 2: Protocol Hardening

```
Wave 1 findings → filed as issues → wave 2 | # | Issue |
Rounds | |-----|-----|-----|
CI-green gate | 2 | | 353 | Parent-session quiescence | 1 | |
351 | γ peer-enumeration | ... | | 350 | Wave primitive | ■ |
| 348 | ROLES.md §4 | ■ | The process fixes itself through
itself.
```



# What CDD is not

## ✗ Not a framework you install.

It's skill files in a repo. No runtime, no SDK, no lock-in.

## ✗ Not multi-agent orchestration.

Roles are sequential, not concurrent. One activation at a time. No message-passing, no persistent agents.

## ✗ Not a replacement for thinking.

Issues still need good ACs. Judgment still matters. CDD just makes it structural.

## ✗ Not all-or-nothing.

Start with  $\alpha+\beta$  on one issue. Add  $\gamma$  when coordination hurts. Add  $\delta$  waves when you need throughput. Earn complexity.

## ✓ A separation of concerns for AI work.

Doing, reviewing, coordinating, operating, and improving — as distinct roles with distinct constraints.

## ✓ Evidence-based quality.

Every approval cites ACs. Every rejection cites findings with line numbers. No "LGTM."

## ✓ Self-improving by design.

The process produces artifacts about its own gaps. Those artifacts become next cycle's work.

## ✓ Protocol-agnostic roles.

Same  $\alpha$ - $\epsilon$  ladder works for code (CDD), prose (CDW), research (CDR), analysis (CDA).

# Getting started

## Your first cycle

- ✓ Pick one issue with clear acceptance criteria
- ✓ Create a cycle branch (`cycle/{N}`)
- ✓  $\alpha$  implements on the branch
- ✓  $\beta$  reviews against the ACs (not taste, not vibes)
- ✓ APPROVED → merge. RC → fix round → re-review.
- ✓ Write what happened (close-out)

## Minimum requirements

- A repo with issues
- Issues with testable acceptance criteria
- Two distinct agents or humans ( $\alpha \neq \beta$ )
- A branch convention

## Scaling up (earned, not imposed)

- **Add  $\gamma$**  when  $\alpha \leftrightarrow \beta$  dispatch overhead grows
- **Add  $\delta$  waves** when you're running 3+ cycles/day
- **Add  $\epsilon$**  when you notice the same failure class twice
- **Automate  $\delta$**  when the dispatch loop is stable

**The role structure is a ladder, not a checklist.** Start at the bottom. Add rungs when evidence demands them. Every role above  $\alpha+\beta$  exists because someone needed it.

Full skill files, worked examples, ROLES.md reference:

[github.com/usurobor/cnos](https://github.com/usurobor/cnos)

# Appendix A — Real cycle: #352 git history

One issue, two review rounds, merged to main. Every commit is a role acting.

```
# git log --oneline cycle/352..main (after merge) dc65c7e5 merge(cdd/352): CI-green gate — β refuses merge on red CI, γ PRA verifies post-merge 3cdec82c β close-out: cycle #352 — CI-green gates delivered and operative ccfb3a77 Merge cycle/352: CI-green gates for β/γ — rule 3.10 + post-merge verification # On the cycle branch (cycle/352): d4d440d5 β R2: APPROVED — CI-green gates complete and operative ← β reviews again cead63f6 α R2: document F1 resolution in self-coherence ← α fixes finding 6f211798 Fix CI-green gate logical gap — enable CI on cycle branches ← α implements fix 96646991 β R1: REQUEST CHANGES — CI-green gate logical gap ← β catches real bug 927d2e9b α: review-readiness signal — ready for β round 1 ← α signals done 3b9fe770 α: self-coherence $Self-check, $Debt, $CDD-Trace b6a07aa7 α: self-coherence $ACs — AC-by-AC evidence mapping 1fe95f74 α: implement CI-green gate rules across 4 CDD skills ← actual work 0a9feef3 α: self-coherence $Skills, $Design, $Plan e3d85b94 α: self-coherence $Gap — CI-green gate rules 27b6520c scaffold(352): γ Phase 1 — cdd-iteration.md + cycle branch ← γ creates scaffold
```

**Every commit names its role and phase.** You can reconstruct who did what from `git log` alone. The commit trail *is* the audit trail.

# Appendix B — $\beta$ catches a real bug

Cycle #352:  $\alpha$  implemented CI-green gate rules.  $\beta$  found the rules were operationally void.

## $\beta$ R1: REQUEST CHANGES

Finding F1 — CI-green gate logical gap Severity: B (binding incoherence) The new rule 3.10 says  $\beta$  must run `gh run list --branch <review-SHA>` and verify CI green before APPROVED. But `.github/workflows/build.yml` only triggers on push to `main`. CI never runs on `cycle/*` branches. The gate is structurally correct but operationally void — the check will always return "no runs found." The cycle that adds CI-green gates cannot verify its own CI.

## $\alpha$ R2: Fix

```
# .github/workflows/build.yml diff on: push: branches: - main + -  
'cycle/*' # One line. The minimum coherent fix. # Now CI runs on  
cycle branches, # making rule 3.10 operative.
```

## $\beta$ R2: APPROVED

Verdict: APPROVED F1 resolved. CI now runs on `cycle/*`. Verified green on review SHA `d4d440d5`. Rule 3.10 is operative — the cycle that ships CI gates passed its own CI gate. Recursive coherence confirmed.

**This is what separation of concerns buys you.**  $\alpha$  wrote correct rules but missed a config gap.  $\beta$  — reviewing with fresh eyes against the ACs — caught that the rules could never fire. A single agent doing both would have shipped void gates.

# Appendix C — The self-coherence artifact

$\alpha$  writes self-coherence.md as it works. This is not a report — it's a live trace of gap  $\rightarrow$  plan  $\rightarrow$  evidence.

```
## $Gap Issue: #352 — CI-green gate Mode: docs-only, design-and-build
CDD protocol has no CI-green verification.  $\beta$  can APPROVE on red CI;  $\gamma$ 
can close-out without verifying post-merge CI status. ## $Plan 4-file
coordination with dependency order: 1. review/SKILL.md —  $\beta$  verdict
gate rule 2. gamma/SKILL.md — post-merge CI step 3. post-
release/SKILL.md — PRA template row 4. release/SKILL.md — §3.8 grade
cap rules ## $ACs AC1 —  $\beta$  refuses APPROVED on red CI. Evidence:
review/SKILL.md rule 3.10 Oracle: rg 'CI.*green.*binding'  $\rightarrow$  hits
Status: ☒ MET AC2 —  $\gamma$  post-merge CI mandatory. Evidence:
gamma/SKILL.md §2.7 Status: ☒ MET AC3 — §3.8 grade caps named.
Evidence: release/SKILL.md §3.8 Status: ☒ MET AC4 — §9.1 trigger
amended. Evidence: post-release/SKILL.md §9.1 Status: ☒ MET AC5 —
Self-applied on this cycle. Status: ☒ MET (R2 verified CI green)
```

## What this artifact does

- $\rightarrow$  **Forces  $\alpha$  to think before coding.** \$Gap and \$Plan are written first — before implementation.
- $\rightarrow$  **Gives  $\beta$  a review anchor.**  $\beta$  walks the ACs in self-coherence.md, checking each oracle against the diff.
- $\rightarrow$  **Creates an audit trail.** Months later, you can read why this change was made and what evidence supported it.
- $\rightarrow$  **Tracks fix rounds.** When  $\beta$  returns RC,  $\alpha$  appends a fix-round section — what was found, what changed, what commit.

Every AC has three parts: **evidence** (what file/line), **oracle** (a grep command anyone can run), and **status**. This makes review mechanical —  $\beta$  runs the oracles and checks the evidence.

## Fix round appendix

```
## Fix Round 1 ( $\beta$  R1  $\rightarrow$   $\alpha$  R2) F1: CI-green gate logical gap
```



# Appendix D — Wave coordination artifacts

δ plans a wave. γ executes each issue. Status is tracked in the repo, not in someone's head.

## Wave manifest

```
# .cdd/waves/hardening-2026-05-12/manifest.md Wave: CDD Protocol
Hardening Date: 2026-05-12 Dispatcher: δ-as-agent | Order | # |
Title | |-----|-----|-----|-----| | 1 | 352 |
CI-green gate | | 2 | 353 | Parent-session quiescence | | 3 | 351
| γ peer-enumeration | | 4 | 350 | Wave primitive | | 5 | 348 |
ROLES.md §4 | Standing permissions: Push to cycle branches: yes
Push merges to main: yes Auto-dispatch α fix rounds: yes (max 3)
Tag/release: NO – operator gate Timeout budgets: γ: 1200s | α:
900s | β: 900s
```

## Wave status (live)

```
# .cdd/waves/hardening-2026-05-12/status.md | # | Issue | Status |
Rounds | |-----|-----|-----|-----| | 352 |
CI-green gate | ✓ done | 2 | | 353 | Parent quiescence | ✓ done
| 1 | | 351 | γ peer-enumeration | ✓ done | 1 | | 350 | Wave
primitive | 🔄 γ | – | | 348 | ROLES.md §4 | 🟡 queue | – |
```

## Iteration findings from wave 1

```
# .cdd/iterations/wave-2026-05-12.md | # | Finding | Sev | |---|-----
-----|-----| | 1 | Wave artifacts in /tmp | B | |
2 | δ not dispatchable | B | | 3 | No wave coordination surface | B |
| 6 | CI failures not caught | B | | 7 | Manual edits → conflicts | C
| → Filed as issues #348-#353 → Became wave 2's work items → The
process improved itself.
```

**Wave 1 ran. It surfaced 7 findings. Those findings became wave 2's issues.** Wave 2 is now fixing the process that wave 1 exposed. This is ε in action — the self-improvement loop with teeth.

# Appendix E — $\beta$ close-out: the review record

After merge,  $\beta$  writes a close-out documenting what happened. This is the cycle's permanent review record.

```
--- cycle: 352 reviewer: beta@cdd.cnos date: 2026-05-12 merge_commit:
ccfb3a77 --- #  $\beta$  Close-out – Cycle #352 Review rounds: 2 R1: REQUEST
CHANGES (1 finding) R2: APPROVED (F1 resolved, CI green) ## Merge
Evidence Branch: cycle/352 Merge commit: ccfb3a77 Merge method: git
merge --no-ff CI status at merge: GREEN Pre-merge gate: Passed Files
merged: .cdd/unreleased/352/* (artifacts) .github/workflows/build.yml
4 CDD skill files ## Implementation Assessment AC coverage: Complete.
All 5 ACs met. AC1: Rule 3.10 –  $\beta$  CI gate ✓ AC2: Post-merge
verification ✓ AC3: $3.8 grade caps ✓ AC4: $9.1 trigger ✓ AC5:
Self-applied ✓ ## Pattern Observed Initial implementation
structurally sound but operationally void – repo config didn't
support rule assumptions.  $\alpha$ 's fix was minimal: adding 'cycle/*' to
workflow triggers.
```

## What makes this useful

- **Permanent record.** Lives in the repo at `.cdd/unreleased/352/beta-closeout.md`. Moves to `.cdd/releases/` on release.
- **Machine-readable metadata.** YAML frontmatter: cycle number, reviewer identity, merge SHA, date.
- **Evidence-linked.** Every AC references files and line numbers. Every finding references specific commits.
- **Pattern extraction.** The "Pattern Observed" section feeds  $\varepsilon$  — if the same pattern appears in 3 close-outs, it becomes a skill patch.

## The full artifact set per cycle

```
.cdd/unreleased/352/ |— self-coherence.md #  $\alpha$ 's trace |— beta-
review.md #  $\beta$ 's verdict(s) |— beta-closeout.md #  $\beta$ 's record |—
cdd-iteration.md #  $\varepsilon$  findings |— alpha-closeout.md #  $\alpha$ 's record
|— gamma-closeout.md #  $\gamma$ 's record
```