TOOLS + AGENTS · ONE SYSTEM

Beyond faster coding: the AI development lifecycle, as one closed loop

Agents brief the code. Shipped code retrains the agents. A dual-model playbook: the coding agent does the engineering, raia agents supply governed knowledge.

raia — the brain
centralized knowledge agents

Coding agent — the hands
plans, builds, tests, ships in-repo

github.com/raia-ai/ai-dlc-playbook

Faster coding alone doesn't deliver faster

24% faster

what experienced developers expected from AI assistance

19% slower

what an early-2025 randomized trial actually observed*

Delivery is a system, not a coding task

Requirements → Design → Build → Test → Release → Operate. Speeding up Build alone just moves the queue: bigger review loads, test pressure, release instability. DORA describes AI as an amplifier — of strong systems and of weak ones.

End-to-end throughput is governed by the slowest constraint.

The hidden tax is waiting

Product → Engineering

ambiguous stories, missing acceptance criteria

Engineering → QA

large changes, thin test evidence

QA → Operations

environment drift, release coordination

Production → Product

signals trapped in separate tools

Fragmented context creates queues — teams wait for clarification, review, access.

*METR study, early 2025 — METR now describes this result as historical, not representative of current tools. Sources: METR 2025; DORA 2025–26.

02 · THE GAP

Two AI revolutions that don't talk to each other

Coding agents transformed the IDE

Claude Code, Cursor, Copilot — they know the repo: plan, implement, review, ship. But they know nothing about your PRDs, policies, or customers.

no learning between them

Knowledge agents transformed the business

raia agents trained on thousands of documents answer support, sales, and ops on every channel. But they know nothing about the codebase — or what shipped yesterday.

What it costs, every week

- Interrupt-driven experts
- Support lags releases
- Docs perpetually behind
- Status by standup
- Slow onboarding

What the PM knows lives in vector stores. What the codebase knows lives in git. Neither learns from the other automatically.

Connect them into one closed loop

Agents brief the code. Shipped code retrains the agents.

raia — the brain

Centralized agents trained on large document volumes in raia Command — reachable from every surface: Copilot, Live Chat, Teams, browser extension, MCP, API.

briefs →

← retrains

Coding agent — the hands

Claude Code or any MCP-capable equivalent: plans, implements, tests, reviews, and ships inside the actual repository — CLI, IDE, and headless CI.

One brain per domain, not per person. The alternative — every tool with its own RAG, every team with its own bot — recreates the silo problem one level down. Tool-agnostic: every play relies only on an MCP client and an instruction file.

Two models, two jobs — by design

Coding agent — execution

Focused on the core task of development. Its context is the repo, the diff, and the session: plan, implement, test, review, ship. The context window stays dedicated to code — not crowded out by PRDs, policies, and help-center exports.

raia agents — knowledge

Support the coding agent with what it can't hold. Each agent searches a knowledge base built from thousands of your documents, and its fixed instructions make every answer grounded, cited, and repeatable — a predictable step in the workflow, not a creative one.

More knowledge than a session can hold

The search runs over the whole knowledge base; only the relevant, already-summarized answer enters the coding session.

Predictable by design

The agent's instructions don't vary by session or developer: same question, same sourced answer — every time.

Each side does what it's best at

Code truth stays in git with the coding agent. Business truth stays managed in raia. Neither crowds out the other.

The coding agent doesn't get smarter by reading more — it gets smarter by asking a system built to know.

Give the AI the right-sized job

TOO BIG

"Build an e-commerce platform."

- Thousands of lines nobody reviewed
- Big design decisions made silently
- Review becomes the bottleneck

THE RIGHT SIZE

A clear goal, with clear limits

- Small, well-defined tasks with a written spec
- The right knowledge on tap — via raia
- Automated checks, then a human sign-off

TOO SMALL

"Write this one function."

- Good output, but barely saves time
- Humans still do all the planning
- Coordination cost barely changes

What makes the right size work: **clear instructions, the right context, connected tools, limited permissions, automated checks, and a human who signs off.** raia supplies the context. The repo supplies the code. Humans make the decisions.

How the two sides connect

RAIA · THE KNOWLEDGE SIDE

Product Agent

Codebase Agent

Support & Docs Agent

Each agent = fixed instructions + a searchable knowledge base of your documents (a vector store), managed in raia Command.

1 · In every session — the coding agent asks, raia answers (via MCP)

2 · After every merge — CI sends updated docs into raia's knowledge bases

3 · For status questions — raia reads GitHub / Jira so PMs can ask "what shipped?"

4 · On events — escalations and merges trigger automations, both ways

YOUR DEV TOOLCHAIN

Coding agent
Claude Code — terminal, IDE, CI

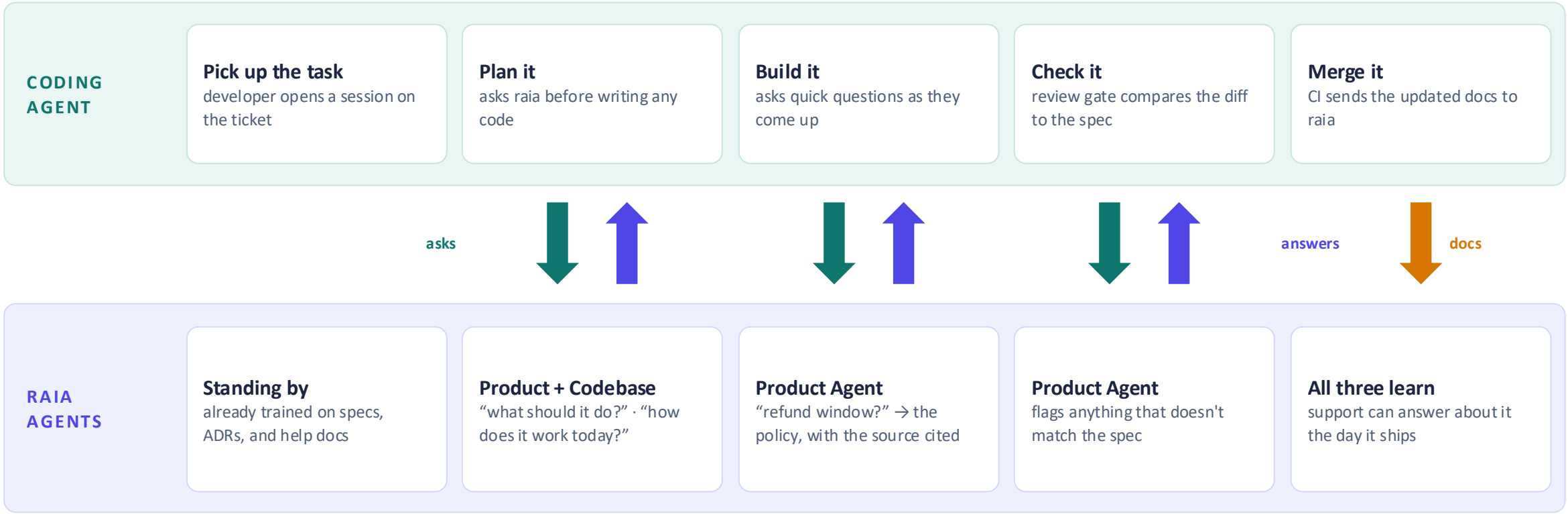
GitHub
repos · issues · pull requests

CI / CD pipeline
runs on every merge

Nothing new to install server-side — the coding agent you already use, connected by config.

MCP is the open standard that lets AI tools call other systems securely. Connection 1 is just two small files committed to the repo — the setup slide shows them; connections 3 and 4 come later in the rollout.

One feature, start to finish — when the coding agent asks raia



Every exchange is a tool call inside the coding session. The developer never leaves the editor, nothing gets pasted between apps — and at the merge, the flow reverses: the code teaches the agents.

raia agents as tools — at every step of the work

PLAN **Before proposing an approach, plan mode interrogates the agents:**

"Plan ticket X. First query raia-product for the spec and constraints, and raia-codebase for prior art and architecture rationale."

BUILD **Mid-task questions route to the right brain instead of a human:**

business rule → raia-product · legacy intent → raia-codebase · real-world usage → support agent

REVIEW **The spec-aware gate closes the loop before merge:**

diff behavior summary → raia-product: "does this match the agreed spec and policies? List mismatches." → PR comments

what it looks like in the terminal

```
> plan the refund-window change
⚙ raia-product · ask("refund window,
  annual plans — policy + criteria")
← "60 days from renewal, POL-114 §2;
  partial refunds prorated. Source:
  Refund Policy v6, updated Mar 2026"
⚙ raia-codebase · ask("prior art:
  refund handling, billing service")
← "BillingService.refund() — see
  ADR-031: idempotency via ledger"
✓ plan cites POL-114 + ADR-031 —
  constraints the dev didn't know
```

Session rules (from the instruction file): agent answers are evidence, not authority — verify against the worktree · never send secrets or raw customer data · stale or missing answer? stop and report the gap, don't invent one.

Three agents cover everything

Launch Pad build the agent, write its fixed instructions → **raia Command** train it: vector stores + connectors → **MCP Skill** expose it as a tool to coding sessions → **Copilot** simulate, supervise, and rate its answers

Codebase Agent

TRAINED ON

Architecture docs, ADRs, generated references, API contracts, migration guides.

ANSWERS

“Why is the code like this?” — for sessions, IDE chats, and new hires.

Product Agent

TRAINED ON

PRDs, strategy docs, acceptance criteria, release notes, pricing & policy rules.

ANSWERS

“What should it do, and why?” — for PMs, plan mode, and spec review.

Support & Docs Agent

TRAINED ON

Help center, docs site, release notes, resolved tickets, known issues.

ANSWERS

Customers in Live Chat, support staff in Copilot, and the weekly triage.

One brain per domain, not per person. Train on documents you already have. Later: front all three with the Orchestrator Skill — one endpoint, one key.

Four steps — and every file ships in the repo

F1

Stand up the agent roster

Build Codebase, Product, and Support & Docs Agents in Launch Pad; train them in raia Command on documents you already have; enable the MCP Skill on each.

F2

Connect the coding agent (per repo)

Commit .mcp.json at the repo root and add the raia section to CLAUDE.md / AGENTS.md — every session in every clone opens already connected.

F3

Connect the IDE

Same server in .vscode/mcp.json for VS Code, Cursor, and Windsurf — verify until logs show "Connection state: Running."

F4

Give PM agents eyes on the toolchain

MCP client mode on the Product Agent pointing at GitHub / Jira — read-only tools, so "what shipped this week?" is answerable in Copilot or Teams.

in github.com/raia-ai/ai-dlc-playbook

`templates/.mcp.json`
the repo connection

`templates/CLAUDE.md-snippet.md`
when to consult raia

`templates/vscode-mcp.json`
the IDE variant

`skills/spec-review/`
the PR review gate

`workflows/knowledge-refresh.yml`
merge → refresh plan

`examples/golden-path/`
credential-free dry run

`docs/quick-starts.md`
first session, per role

Prove it before connecting anything live: `cd examples/golden-path && npm test && npm run dry-run` — the full evidence chain, no credentials required.

Build the agents to complement the coding tool

Fixed instructions — how it answers

Written once in Launch Pad; identical in every session, for every developer:

- Answer only from retrieved sources — cite document and section
- If it isn't in the knowledge base, say so; never improvise an answer
- Stay in scope: one domain per agent; route other questions onward
- Return synthesized answers — never raw transcripts or customer data

The knowledge base — what it knows

Vector stores in raia Command, curated so answers stay accurate as content grows:

- One corpus per audience — route content by explicit policy, never guesswork
- Author knowledge as Markdown near the code: specs, ADRs, docs in the repo
- Delete-and-replace on every refresh, so retrieval never surfaces stale versions
- Verify freshness after refresh: ask the question, check the retrieval, file gaps

The result: the coding agent gets a reliable answer service, not a chat partner — **the instructions control how it answers; the knowledge base controls what it knows.** Both are versioned, reviewed, and owned.

Where to engage the agents — phase by phase

1 · Ground	**PM · Copilot**	Ask the Support & Docs Agent: “top friction points related to X, with example conversations” — the evidence note links from the spec.
2 · Specify · the PRD	**PM · Copilot / Chat**	Draft the PRD with the Product Agent — it flags overlap with existing features and reuses acceptance-criteria patterns. Commit the spec as Markdown in docs/specs/.
3 · Plan	**Developer · plan mode**	The coding agent queries raia-product (spec + constraints) and raia-codebase (prior art, rationale) before proposing an approach.
4 · Build	**Developer · in-session**	Route mid-task questions: business rule → Product Agent · legacy intent → Codebase Agent · real-world usage → Support Agent.
5 · Review	**Automated · PR gate**	The spec-review skill sends the diff's behavior summary to the Product Agent — confirmed mismatches land as PR comments.
6 · Refresh · the docs	**Coding agent + CI**	Docs and changelog drafted from the diff in the same PR; on merge, changed Markdown routes to the right agent's vector store.
7 · Verify	**Support lead · Copilot**	Simulations ask about the newly shipped behavior; Admin Mode confirms fresh retrieval — gaps become docs tickets, not chat threads.

Highlighted rows are where knowledge crosses the loop: the PRD is drafted with the agents, and the documentation refresh teaches them what shipped. The pilot succeeds when one real change completes all seven.

Twelve plays across three tracks — start with four

TRACK A · BUILD

A2 · Agent-briefed planning

plan mode interrogates Product + Codebase Agents before proposing an approach

A3 · Questions stay in-session

business rule → Product · legacy intent → Codebase · usage → Support

A4 · Spec-aware review gate

diff summarized, checked against the spec; mismatches land as PR comments

TRACK B · SUPPORT

B1 · Day-one readiness

merge retrained the Support Agent; lead verifies with Copilot simulations

B2 · Escalation-to-issue pipeline

Live Chat bug → structured GitHub issue; narrow classes get a draft PR

B4 · Internal answer desk

sales, CS, leadership query the same agents — one truth, many doors

TRACK C · DOCUMENTATION

C1 · Merge = retrain (keystone)

CI routes changed docs to the right agent's vector store, by audience

C2 · Docs written by the diff

the coding agent drafts docs + changelog in the same PR as the code

C3 · Gap-driven backlog

failed retrievals become docs tickets — the same question never fails twice

Pilot = A2 + A4 + C1 + B1 in one repository. Highlighted borders mark the starting four. Full play cards, templates, the spec-review skill, and the C1 workflow: github.com/raia-ai/ai-dlc-playbook

The slide that answers the security question

Scoped keys, rotated

Per-agent keys in env vars and CI secrets, never committed. A leak affects one agent; rotate from Launch Pad.

Read-only by default

Conversational agents get read tools only. Write access lives in audited automations — and is proven in dry-run first.

PII stays governed

Customer data remains in raia's stores — retention controls, immutable logs. Coding sessions get synthesized answers, never transcripts.

Humans gate every merge

Automation drafts PRs; it never merges them. The allowlisted class grows only with demonstrated merge rate.

Fail closed: ambiguous routing, stale state, missing review, or failed verification stops the workflow. Nothing gives an AI u nattended write access to production code or customer data.

Measure outcomes — not generated output

Time-to-context

minutes from “assigned” to credible plan

falls sharply

Knowledge lag

merge → agents answer correctly about it

same day

Escalation quality

support bugs with usable repro steps

≥ 80%

Retrieval failures

agent queries with no relevant retrieval

declines monthly

Interrupt load

ad-hoc pings landing on engineers

migrates to agents

Mismatch catch point

where spec deviations surface

PR, not UAT

Retire as a KPI: lines of code generated. **Adopt as the question:** are we delivering better software faster, with less risk and less friction? Baseline before you target; review at day 90 and kill any play that doesn't move its metric.

30 / 60 / 90 — configuration before construction

DAYS 1–30 · FOUNDATION

Build the three agents on existing docs. Commit MCP config + instruction files. Run one real change through all seven steps of the loop.

DAYS 31–60 · HABITS

Agent-briefed planning and docs-in-the-PR become convention. C1 knowledge refresh in all repos. Weekly triage and gap review.

DAYS 61–90 · AUTOMATION

Spec-aware gate on every PR. Escalation pipeline live, draft-PR class piloted. Day-90 metrics review: widen or kill each play.

The ask

1. Approve the 30-day foundation — three agents, config in every repo, one CI job
2. Nominate four owners — developer champion, PM, support lead, docs owner
3. Book the day-90 metrics review — widen what works, kill what doesn't

The playbook, templates, workflow & skill:

`github.com/raia-ai/ai-dlc-playbook`

Every merge makes the whole company smarter.