From Coder to Coach

Working with AI Agents in Open Source Development

Guillaume Nodet
March 2026

The Confession

I raised **dozens of PRs** on Apache Camel in the past two weeks.

All written by AI agents.

Some were great.

Some were **really wrong**.

Here's what I learned.

The Shift

What does the developer role look like now?

From Writing Code to Leading Agents

The hands-on-keyboard coding part is shrinking.

What grows:

- **Breaking down problems** and giving clear direction
- **Judging code quality** and correctness
- **Managing parallel workstreams** — multiple agents, multiple PRs
- **Architectural decisions** — the stuff AI can't figure out alone

Think of it as becoming **team lead for a team of AI agents**.

The Toolkit

Claude Code

CLI-based AI coding agent

- Reads code, runs commands, edits files
- Interacts with GitHub and JIRA APIs
- `/effort max` – deep reasoning mode
- `CLAUDE.md` – persistent instructions
- Custom commands – encode lessons into reusable workflows

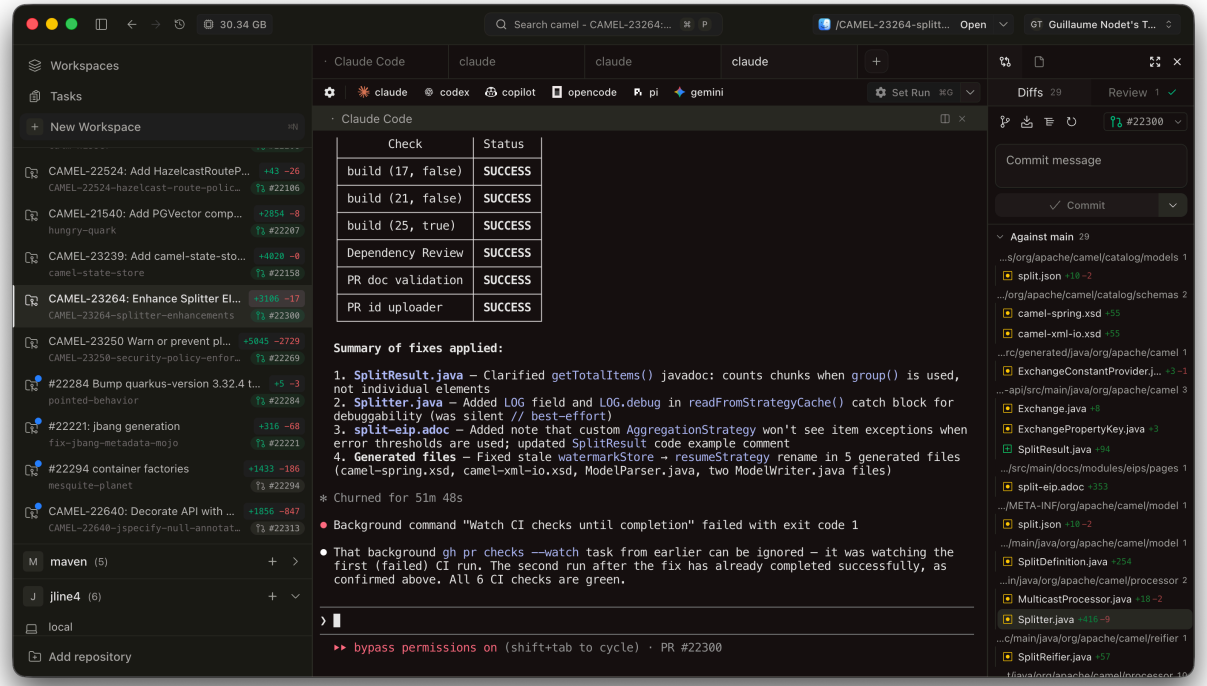
github.com/gnodet/claude-code-commands

Superset

Terminal workspace manager on git worktrees

- Each workspace = branch + directory + agent
- Parallel workstreams, no `git stash` chaos
- One task per workspace, full isolation

Superset in Action



What Goes Wrong

The three rules I learned the hard way

Rule 1: High Effort Mode for Planning

Without `/effort max`, the agent's reasoning is **too shallow** for non-trivial tasks.

- Skips edge cases
- Misses architectural constraints
- Picks the **obvious-but-wrong** approach
- Plans look reasonable… until you read the code

Fix: Always use `/effort max` before any planning step. The difference is night and day.

Effort Max — Practical Tip

The `effortLevel: "max"` setting in `settings.json` **doesn't reliably work**.

Workaround: use the CLI flag directly.

```
# Shell alias
alias claude='command claude --effort max'

# Or for Superset: patch ~/.superset/bin/claude
exec "$REAL_BIN" --effort max "$@"
```

Known issue — fix in progress.

Rule 2: Make the Agent Gather Context

The agent doesn't know what it doesn't know.

- Prior discussions on related JIRA issues
- **Intentional** design decisions in git history
- Why code was written a certain way (`git blame`)
- Related tickets, rejected approaches, design docs

Fix: Encode it in `CLAUDE.md` . I now require the agent to:

```
Before implementing, you MUST:
- Check git history and git blame on affected files
- Search for related JIRA tickets
- Read commit messages for prior changes
- Look for design documents in proposals/
```

I've since formalized this into reusable **custom commands** (`/work-on` , `/review-pr` , etc.).

Rule 3: Don’t Trust – The Tunnel Problem

When an agent commits to a wrong approach, **it doesn’t self-correct**.

It doubles down. It keeps writing code that’s internally consistent but **fundamentally wrong**.

Like a junior dev who’s too confident – they won’t stop and question their assumptions.

Fix:

- Review the **plan** before implementation
- Question assumptions early
- If something feels off, **stop and re-evaluate** – don’t let the agent dig deeper

This is where **agent swarms** could help – but not just for cross-checking (see next).

What Works

When you follow the rules, here's what's possible

Example: The Full Loop

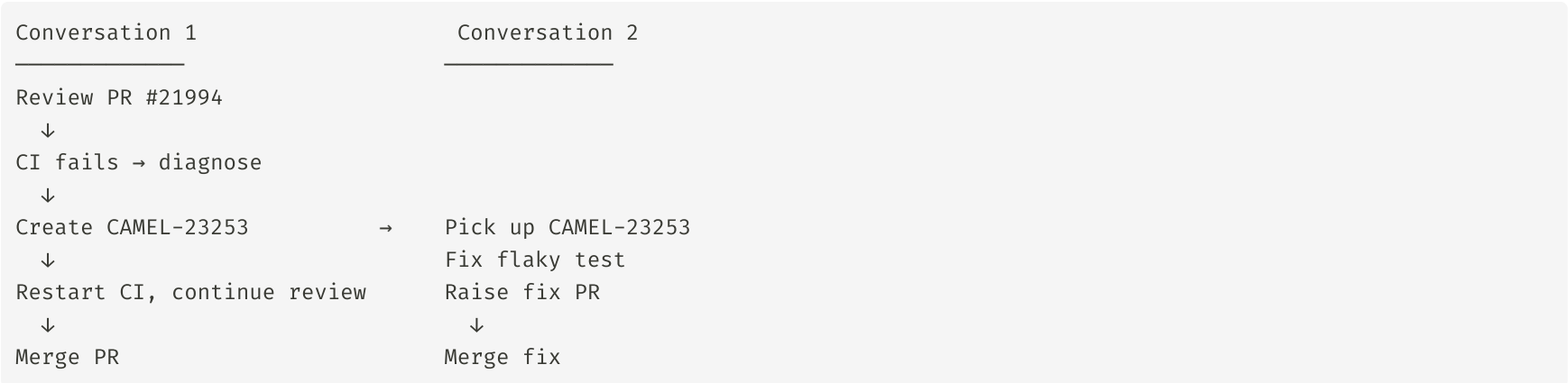
While reviewing PR **#21994**, I asked the agent to:

1. Check if all review concerns had been addressed
2. Analyze why CI failed on the latest build
3. The agent downloaded CI logs, identified a flaky Infinispan test — **unrelated to the PR**
4. I asked it to check if a JIRA already existed — it found nothing
5. I told it to create one → **CAMEL-23253**
6. I had it restart the failed CI job

One conversation. Review + triage + issue creation + CI restart.

My role: **reviewer** and **coordinator**. Not a single line of code written.

The Multiplier Effect



Two parallel workstreams. One human. Zero context-switching cost.

My role: **orchestrator**. I decide what to delegate, when to fork, and review the results.

PR Babysitting

After pushing a PR, the iteration loop:

- **CI fails?** → I tell the agent to diagnose → it downloads logs, finds root cause, fixes, pushes
- **Review comments?** – two kinds:
  - **Code fixes / suggestions** → "address the review comments" → agent fixes code, pushes a commit
  - **Questions / remarks** → I discuss with the agent, then ask it to post the answer
- **Formatting / artifacts** → agent runs formatter, regenerates, pushes
- **Repeat** until CI is green and reviewers are satisfied

The "push → wait → check email → context-switch → write fix → push" loop becomes **"tell the agent what to fix → review → push"**.

I still direct every step – but I never write the code myself.

JIRA Triage

Finding the next thing to work on:

- Query JIRA for unassigned issues in a component
- **Cross-reference with the codebase** — grep for related code, check git history
- Assess complexity, impact, whether the issue is still valid
- Flag issues that are duplicated or already fixed
- Present a **curated shortlist** with context already gathered

Replaces opening dozens of JIRA tabs and manually jumping into code for each one.

My role: **architect**. I make the strategic call on what's worth working on.

What’s Next

The Road Ahead (1/2)

Agent Swarms — Fighting Context Pollution

As the context window fills up, the agent **forgets the rules** from the start of the conversation. The literature calls this **context pollution**.

Claude Code already spawns sub-agents with isolated context — but the main agent still accumulates, and sub-agents delegate, they don't cross-check. Next step: independent agents verifying each other's work.

Automatic PR Code Review

CodeRabbit — free for open source, reads `CLAUDE.md`, configurable. Alternative: a custom GitHub bot delegating to a Claude instance with full project context. Both need investigation.

The Road Ahead (2/2)

Self-Spawning Workspaces

Agent encounters a side issue → creates JIRA ticket → spins up a new workspace → kicks off a parallel agent. Just a human confirmation to proceed.

The Developer as Team Lead

One human managing a small team of AI agents, each working on a different task, all in parallel. Review the PRs, make the decisions, ship the code.

The developers who thrive will be the ones who learn to be effective team leads for AI agents.

Know when to trust. Know when to verify. Know when to step in.

Full write-up & practical tips: **blog.gnodet.fr/posts/ai-workflow-presentation**

Guillaume Nodet – @gnodet