

You are the engineer. And the team lead.

One real feature: an AI assistant that resolves support tickets when appropriate.

AI-TDD

Which scenarios must pass?

Design-oriented implementation

How should the solution be designed?

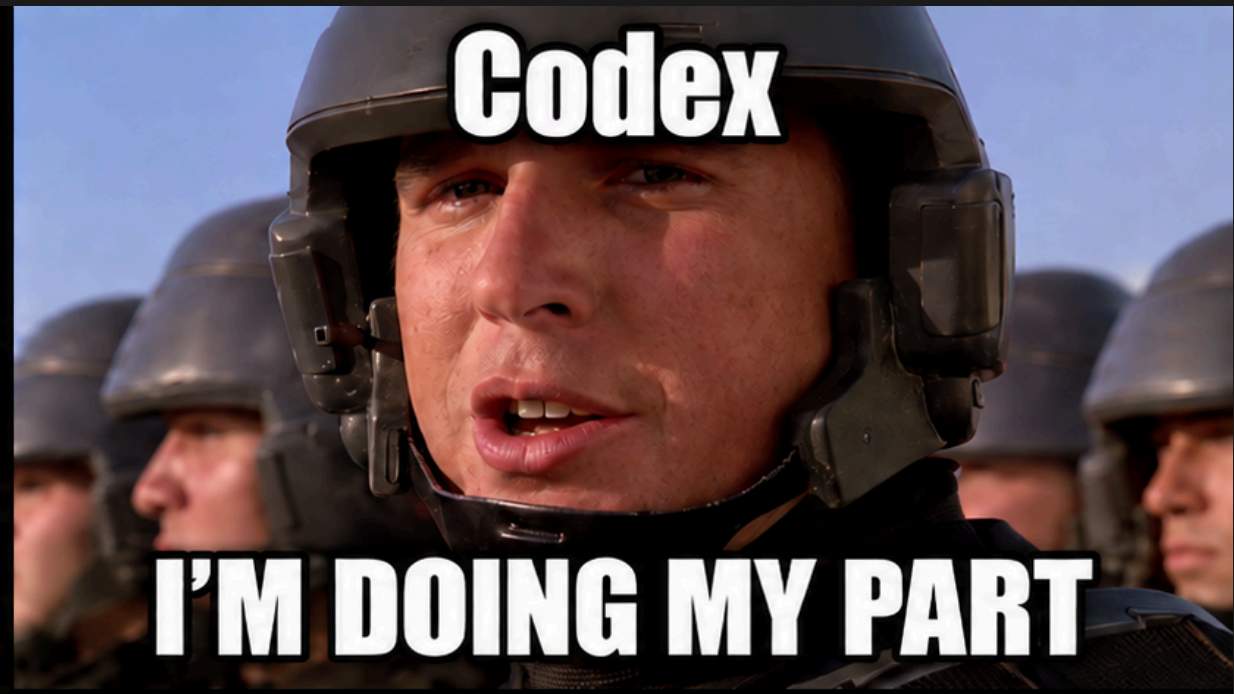
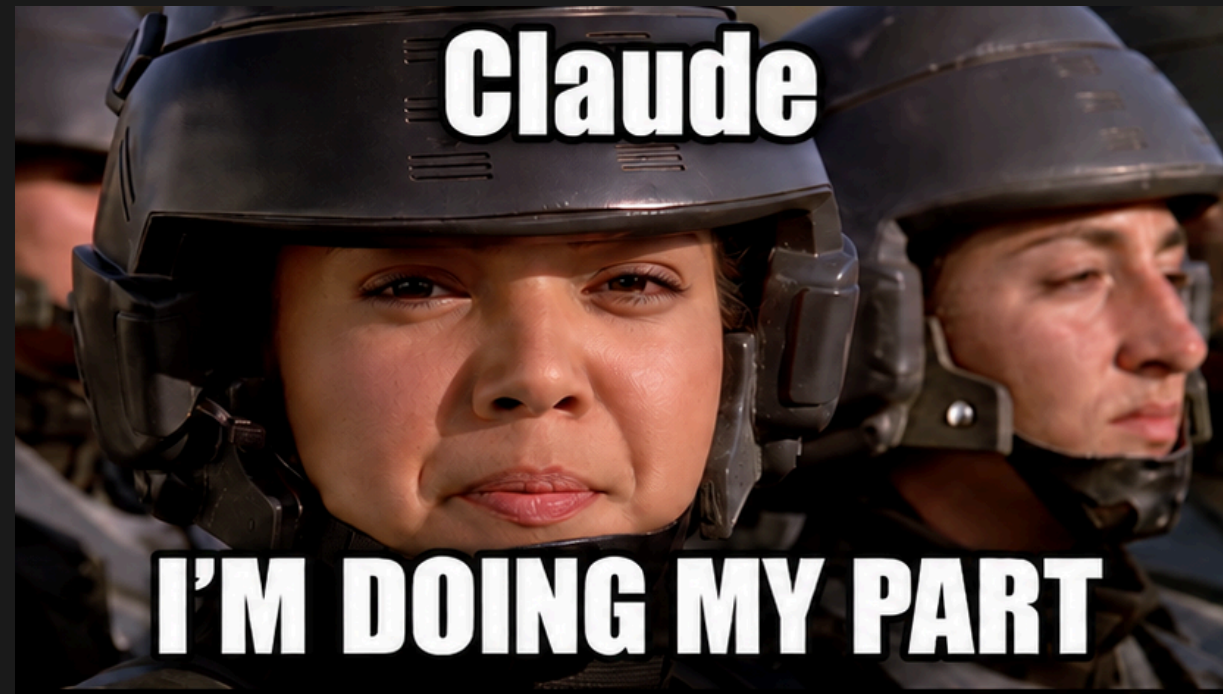
Business-oriented mindset

What should improve for the customer?

Agentic Mode in Action

How does an AI team deliver it?

I'm doing my part



“Could the assistant resolve this itself?”

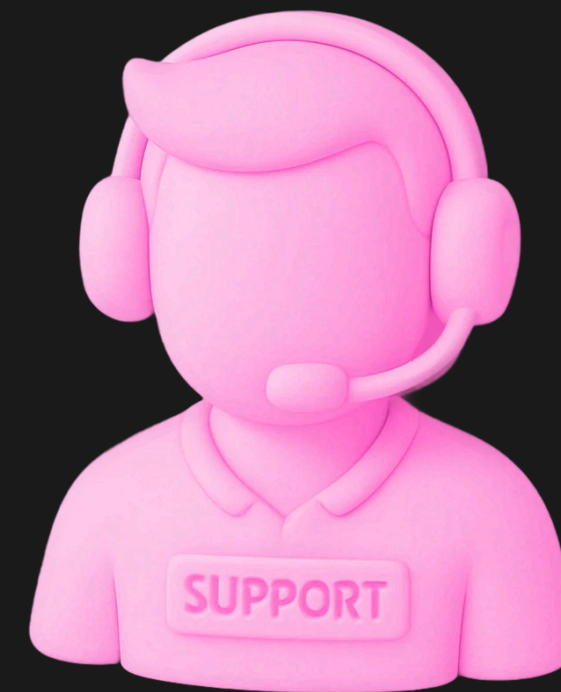
THE BUSINESS REQUEST

Resolve tickets when it is appropriate.

The value: less manual support work and fewer solved tickets left open.

The hidden work: decide what counts as resolved, when to ask, when to wait and what the customer sees.

Most of the difficult decisions describe business behavior.



“It’s just a close button... right?”

AI - TDD

We agree on E2E scenarios and regressions **before implementation**. After each meaningful iteration, we run them again.

In ticket closure

“Please close it” and “Thanks” are different scenarios with different expected outcomes.

What changes for the engineer

You define observable success before reviewing the generated solution.

The evidence is the ticket state and customer experience, not the assistant saying “done”.

One change. The same checks. Every iteration.

01

Agree scenarios

Customer actions and expected outcomes



02

Implement

One coherent change



03

Run E2E + regression

Check UI, state and effects



04

Inspect evidence

Pass, fail or not run

← On failure or the next change, repeat the cycle

"Please close it"

Resolve once

"Thanks"

Confirm before closure

"Not yet"

Keep the ticket open

Retries, concurrency and failure paths belong in the same acceptance suite.

Design-oriented implementation

As automation raises the level of abstraction, we spend less effort writing routine code and more effort **designing code people can understand**.

What we delegate

Validators, conditionals, mapping code and helper methods within an agreed design.

What we focus on

Clear responsibilities, readable flows, component contracts and architecture.

The goal: understand the feature's main flow and failure paths in a five-minute first read.

Compiler: source becomes executable

1. Compiler → 2. Declarative generation → 3. AI-assisted design

WE DESCRIBE THE PROGRAM

```
int add(int a, int b) {  
    return a + b;  
}
```

We express behavior in a source language.

THE TOOLCHAIN PRODUCES THE EXECUTABLE



Compile

Target instructions

Object code

Linked executable

Compilation, assembly and linking handle the translation.

Routine development moves to source code. We inspect lower-level output when the task requires it.

The abstraction changes where we work. We still test the program and investigate failures.

MapStruct: mapping rules become Java

1. Compiler → 2. Declarative generation → 3. AI-assisted design

WE DECLARE THE MAPPING

```
@Mapper
interface TicketMapper {
    @Mapping(target = "title",
              source = "subject")
    TicketView toView(Ticket ticket);
}
```



Generate
at build
time

GENERATED SHAPE · SIMPLIFIED

```
public TicketView toView(Ticket ticket) {
    if (ticket == null) return null;

    TicketView view = new TicketView();
    view.setTitle(ticket.getSubject());
    view.setStatus(ticket.getStatus());
    return view;
}
```

One explicit field rename. Matching properties map by convention.

The processor supplies the null check and property assignments.

The engineer owns the mapping contract. The generator supplies repetitive Java.

Compile-time annotation processing, not runtime AOP. Example assumes compatible bean properties.

And here comes AI



HERE'S JOHNNY!!!

AI: design guides implementation

1. Compiler → 2. Declarative generation → 3. AI-assisted design

OUR DESIGN INSTRUCTIONS

Separate responsibilities. Keep intent interpretation out of persistence.

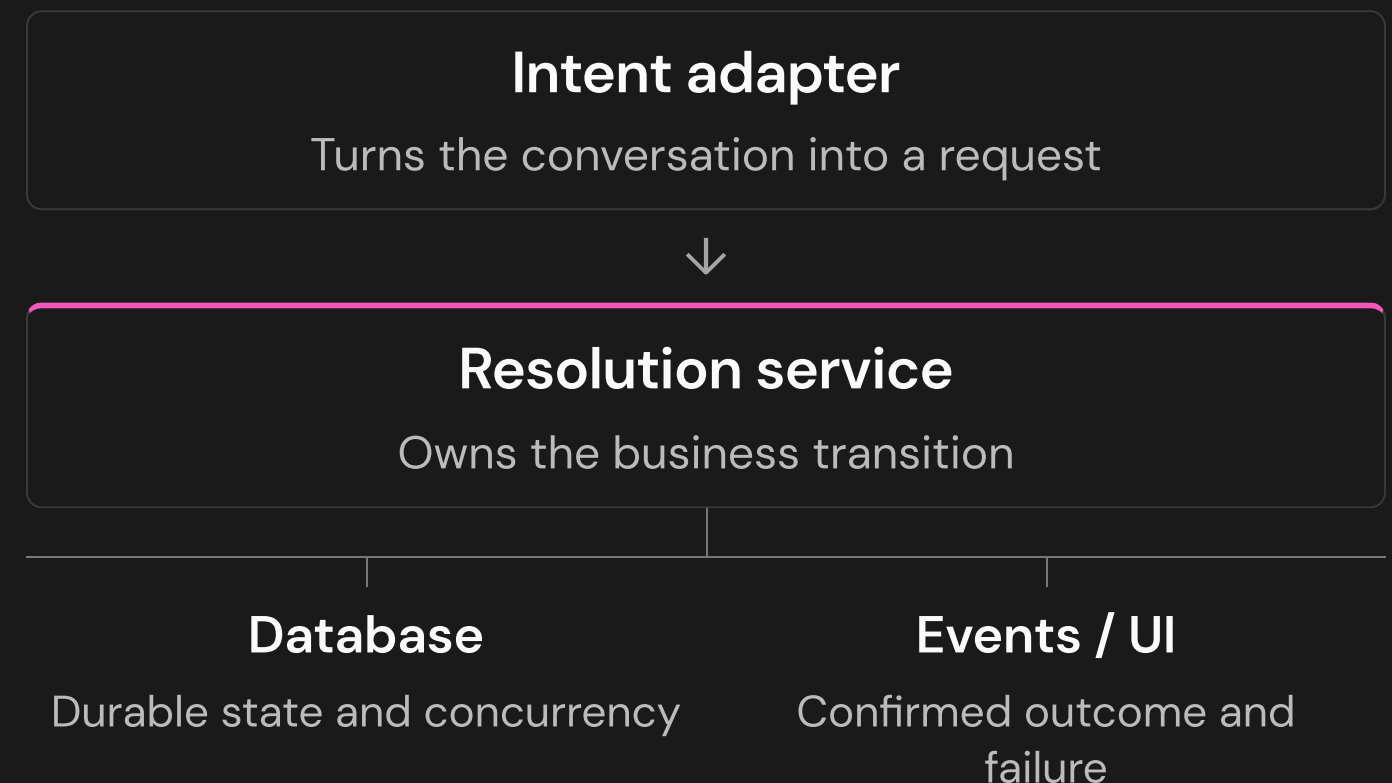
Make the flow readable. Use explicit contracts and named business steps.

Delegate routine code. Generate validators, helpers and mappings within those rules.

Can a teammate explain the main closure flow after a five-minute first read?

| We still own concurrency, database safety, logs, performance and edge cases.

AI interprets instructions. Unlike a compiler or a fixed generator, its output requires explicit review and verification.



Business-oriented mindset

Technical correctness is the baseline. Our contribution grows when we **anticipate what the business and its customers will need next.**

Implementation focus

“Does the code work?”

Latency, $O(n)$ vs $O(\log n)$, null safety and unit tests.

Business-oriented focus

“Does the feature solve the customer's problem?”

Real usage, missing scenarios and the next useful capability.

We help shape the product. We share responsibility for the business outcome.

Vulnerability search: the next questions

The request: “Show vulnerabilities on this device. Find a specific CVE.”

The literal implementation can return a matching list. The business question is whether the customer can use it.

What might the customer need next?

What we should discuss before implementation

More than 50 results

Pagination and a clear total. No silent truncation.

Help deciding what to fix first

Classification and filters, such as severity.

The same CVE across all devices

Search across the fleet and show affected devices.

All vulnerabilities across all devices

Clarify the scope, summarize and offer narrower filters.

Think several steps ahead. Agree what belongs in this release and keep the design open to the next step.

The code works. Can the customer act?



Correct execution is the baseline. Helping the customer decide is the next responsibility.

Agentic Mode in Action

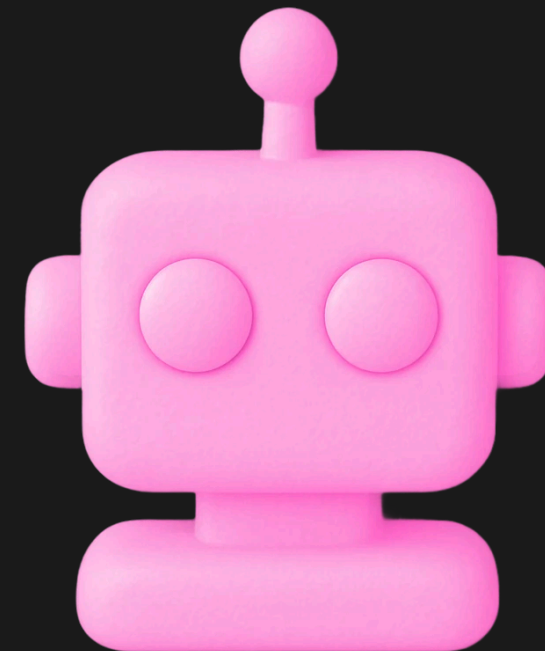
THE SAME TICKET. A WORKING TEAM.

**Separate contexts.
Explicit
responsibilities.**

Agents investigate, implement, test and review bounded parts of the task.

You lead the business decisions and approve the design.

Now we move from principles to configuration, delegation and real execution.



Useful teammates need a clear assignment.

A team shares evidence, decisions and feedback



Conceptual team illustration. The main agent coordinates the work; you own the goal and approvals.

A role needs a concrete output

Agent	What you should receive
Business analyst	Scenarios, scope and unanswered product decisions
Investigator	Evidence, affected repositories, effort and risks
Implementer	A focused diff within the approved plan
Tester	Passed, failed and untested scenarios with evidence
Reviewer	Prioritized race, data–safety, compatibility and design findings

Optional environment specialist: Git, local tooling, Telepresence, CI and deployments.

A real team investigating an AOP refactor

Stage 1 draft brief: target is `openframe-saas-tenant/openframe/services/openframe-saas-ai-agent` ; goal is behavior-preserving refactoring that moves genuinely repeated cross-cutting processing into Spring AOP. Ticket, exact acceptance criteria, and the specific repeated flows are still pending. I'm starting parallel read-only architecture, feasibility, and worktree-topology checks.

🌐 ✨ 🔄 Ai aop investigation, Ai aop grooming and Ai aop worktree plan started working

📁 Read files, ran commands

🔄 Ai aop worktree plan finished

📁 Read files, ran commands

Stage 2 is narrowing the draft: the first evidence-backed AOP candidate is notification failure isolation—15 repeated “catch, log, swallow” entrypoints across five dispatchers. Streaming lifecycle, token billing, tool approvals, ticket transitions, and lock handling are not safe to hide in synchronous aspects because their ordering changes observable behavior.

✨ 🔄 Ai aop grooming and Draft progress finished

Команда завершила оценку реализуемости и планирование, сейчас исследует архитектуру рефакторинга AI-сервиса. Изменений кода и развёртывания нет; ожидаются тикет и критерии приёмки.

🌐 Ai aop investigation finished

01

Parallel assignments

Architecture, feasibility and worktree topology.

02

Returned findings

A candidate refactor and explicit risks.

03

Visible status

No code changes or deployment. Ticket still pending.

Actual discovery run, separate from our ticket-closure example. Click the screenshot to inspect the original.

Communication between agents

- Sent message to Memory state lib refactor
- Sent message to Remove memory activation lib
- 🔗 Reading Code Style skill
- 🌀 💠 Memory main merge preflight and Remove memory activation lib started working

Messages go to specific teammates. The activity log shows which tasks start working.

Agent configuration: Claude and Codex

Same reviewer responsibility. Different file formats.

Claude · `.claude/agents/reviewer.md`

```
---
name: reviewer
description: Review completed changes.
tools: Read, Grep, Glob
---
Check races, database safety,
compatibility and acceptance criteria.
Return prioritized findings and evidence.
Do not modify files.
```

Codex · `.codex/agents/reviewer.toml`

```
name = "reviewer"
description = "Review completed changes."
sandbox_mode = "read-only"
developer_instructions = """
Check races, database safety,
compatibility and acceptance criteria.
Return prioritized findings and evidence.
"""
```

The role file defines responsibility. Repository instructions define the working rules.

How the team starts

CLAUDE · `.claude/settings.json`

```
{
  "env": {
    "CLAUDE_CODE_EXPERIMENTAL_AGENT_TEAMS":
      "1"
  }
}
```

Agent Teams is experimental. Start an interactive session, then request the team.

Codex routing lives in applicable AGENTS.md instructions or workflow skills. Human approval gates must be explicit.

01 Enable the capability

Verify agent tools in the installed version.

02 Define reusable roles

Give each role tools, boundaries and an output.

03 Add routing instructions

For a large ticket, investigate first and return a plan before editing.

The team brings evidence. You decide.

Parallel work can reduce waiting. **You own the decisions and coordination costs.**

Decision point	What the team brings	What you decide
Before implementation	Scope, design, effort and test scenarios	Does the plan solve the business need?
Before accepting the result	Diff, regression evidence and review findings	Accept, request changes or investigate further?

Where parallel work helps

Independent investigation, test preparation and review.
Less waiting and less unrelated context per agent.

What the lead must coordinate

Shared files, changing contracts and conflicting assumptions. More tokens, duplicate work and handoff risks.

Implementation approval does not automatically authorize commits, pushes, PR changes or deployment.

A failed check needs a decision

Illustrative ticket-closure review. These rows are examples, not results from a real test run.

#	Scenario	Why the check failed	Investigator comment	Proposed fix / next step
1	Retry a closure request	Two closure notifications instead of one	Duplicate notification on retry.	Fix: prevent duplicate notification. Rerun retry checks.
2	Customer says "Thanks"	Test expected closure. Ticket stays open.	Policy requires confirmation. Auto-closure would break it.	Keep behavior: correct the test expectation.
3	Confirm closure, inspect UI	Ticket still appears open when the check runs	Could be UI delay, failed closure or an early check.	Investigate: check saved state and UI timing.

The team lead decides: fix a defect, preserve business behavior or gather more evidence.

An engineer who leads an AI team

YOUR NEXT TICKET

You still own the outcome.

AI-TDD: agree the evidence.

Design-oriented implementation: choose the boundaries.

Business-oriented mindset: solve the right need.

Agentic Mode: delegate focused execution.

Start with one ticket. Review the decisions as carefully as the code.



The team got bigger. So did your responsibility.

Hooks: checks before the action

The idea overall is to perform refactoring around AI service in order to hide redundant processes into AOP instruments. Use agentic team instructions. Use work tree.

Worked for 24s >

Please send the ticket URL or paste its full content, including acceptance criteria.

The team flow requires the ticket before discovery. I have not created a branch/worktree or changed code. After discovery and your plan approval, implementation will use

```
/Users/admin/Downloads/projects/.worktrees/<repo>--<topic> .
```

Observed behavior: asks for a ticket before discovery. The screenshot alone does not establish whether a hook enforced it.

ENFORCEMENT IDEA

No approved reference? Block the covered action.

Instructions and skills tell the model what to do.

A blocking code hook checks a condition before a supported tool call executes.

Task gate · ticket or approved spec

Git policy · checkout, branch, commit approval

Environment policy · production target or flag change

Hooks cover defined execution paths, not every possible action. Protect production with permissions and server-side controls too.

| Interested? We can cover hooks and other practical controls in a separate session.