WORKSHOP CANVAS

# AI-Native Engineering Canvas.

A workshop template for designing how your team adopts AI-native engineering: an operating model, not a tool checklist. Together the eight areas describe the harness your team builds around its agents, the context, tools, checks, and feedback loops that make AI-assisted work safe and fast.

## HOW TO USE THIS CANVAS

Export this page to PDF or screenshot it into **Miro**, or just print it. Then go area by area and answer the prompts *together*, using sticky notes for what is already in place and what is still missing. Around 60 to 90 minutes the first time; much shorter on later passes.

It also works **async**: drop it in a repo doc or a Slack thread, let everyone annotate on their own time, then meet only to resolve the points you disagree on. The value is the shared vocabulary and the disagreements it surfaces, not the meeting itself.

## QUESTIONS FROM THE TEAM

### What do we actually walk away with?

A short list of **decisions and open disagreements**, not a poster. The real output belongs *in the codebase*: agreed conventions go into `AGENTS.md` or tool configs, specs into your specs folder, and every unresolved point becomes a ticket.

### Isn't this just deciding everything upfront?

Only if you fill it with wishes. Answer *only from what you already do*; leave everything else blank. A **blank is signal**, not failure. It marks something the team hasn't figured out yet and should stay open. You're describing reality and finding its gaps, not committing to a plan.

### When do we use it: start, or mid-project?

**Both.** Run it once when a project starts, to get aligned before habits set. Then revisit it *lightly* every few months, or whenever something changes: a new model generation, a practice that didn't work out, a tool you dropped. It's a checkpoint triggered by change, *not* a calendar ritual.

### Is this overkill for a senior team?

It can be. If practices already surface cleanly in *standups and PR reviews*, you may not need this at all. It earns its keep when seniority is **mixed**, or when *strong opinions collide* in a field with no historical baseline. AI-native work has plenty of those grey areas, and this makes the implicit choices explicit before they harden into arbitrary rules.

## 01 Team & north star

The team, system, and constraints, and why we're going AI-native.

- What product or system does the team own, and is it greenfield, brownfield, or mixed?
- Which parts of the codebase change most often, and where would a bad change do the most damage?
- What constraints shape the work: compliance, security, uptime, budget, deadlines?
- What outcome are we optimising for, and which bottleneck are we removing?
- What should be visibly better in 30, 60, 90 days?
- What must not get worse while we adopt AI?

## 02 AI-native use cases

Where agents help today, and where they don't yet.

- Which tasks are well-scoped and easy to verify?
- Which are repetitive and low-risk?
- What can we fully automate with agents running in the background?
- Which need human judgement before any code?
- Which should use AI only for drafting, review, or analysis?
- What stays fully human for now?

## 03 Roles & accountability

Who owns intent, review, and production outcomes.

- Who owns the problem definition and approves the spec before work starts?
- Who reviews AI-generated code and decides a change is safe to ship?
- Which decisions must never be delegated?
- Who owns incidents caused by AI-assisted work?
- When should the agent stop and ask for help?

## 04 Context & specs

What agents need to know, and how we write intent before implementation.

- What context is needed to work safely, and where does it live today?
- What is stale, missing, or tribal knowledge?
- What rules, style guides, or examples should agents reuse?
- What skills might we need?
- Which work needs a written spec first?
- Who reviews specs, and how short should they be?

## 05 Tooling

The agents, MCP servers, CLIs, and skills we trust.

- Which tools are approved for code, data, and client work?
- MCP server, or CLI script / skill instead?
- How do we avoid too many tools and context pollution?
- What credentials or permissions do tools need?

## 06 Verification & delivery safety

How we prove work is correct and get it to production safely.

- What deterministic checks must pass before merge?
- When should an LLM reviewer be used, and for what?
- Which checks block merge, and which only warn?
- What needs feature flags, canary, or staged rollout?
- What telemetry must exist before release, and what is the rollback process?
- How do production incidents feed back into specs and tests?

## 07 Security & governance

The boundaries that keep tools safe.

- What data can and can't be shared with AI tools?
- What permissions should each tool have?
- What approval is required before destructive actions?
- How are secrets protected?
- Who approves new tools, and how do we audit tool use?

## 08 Metrics & learning loop

How we know it's working, and how the system improves after every change.

- How do we measure cycle time and change-failure rate?
- How do we track AI cost, latency, and developer experience?
- What metric would tell us to slow down?
- What do we update after each change ships: rules, specs, prompts, skills?
- How do incidents become regression tests?
- How do we stop context and docs from rotting, and onboard new members?