

THE AI-ASSISTED DEVELOPER PLAYBOOK



How We Rewired Our Engineering Workflow
with AI — And What We Learned

AIEnterprise Inc



THE UNCOMFORTABLE TRUTH

Most developers use AI like autocomplete.

We use it like a **second brain**.

The gap between "AI-curious" and "AI-native" engineering teams is no longer about tools. It's about workflow design, context architecture, and knowing when to lead vs. when to delegate.

THE MINDSET SHIFT

From writing code to directing outcomes

~~Single-Role Developer~~ → **Multi-Role Operator**

One developer now plans, architects, decomposes, and implements — with AI handling the grunt work.

~~Manual Research~~ → **Guided Discovery**

30-minute code spelunking sessions become 2-minute AI-guided walkthroughs. You get 80% context instantly.

~~Isolated Coding~~ → **Contextual Collaboration**

Your codebase context files become a shared brain. AI compounds knowledge across every session.

THE 7-STEP DAILY WORKFLOW

A repeatable recipe for every task, every day

01 Understand

AI reads the ticket + codebase, produces a plan

02 Dependencies

AI flags cross-repo impacts you'd miss

03 Decompose

Break into subtasks — AI or you, depending on context

04 Implement

Agent mode generates pattern-following code

05 Review

You are the quality gate, always

06 Test

AI scaffolds tests; you validate correctness

07 Compound

Update context files — make next session smarter

CONTEXT IS THE MOAT

The single biggest lever for AI effectiveness isn't the model — it's what you feed it.

CLAUDE.md — Your AI's Briefing Doc

- Project purpose & scope
- Tech stack & architecture
- Coding conventions & patterns
- Cross-repo dependencies
- Known quirks & tech debt

THE COMPOUNDING EFFECT

Better context files

→ Smarter AI output

→ Faster delivery

→ More learnings captured

→ Even better context

→ **Repeat. Compound. Win.**



PROMPTING IS A SKILL, NOT A GIMMICK

VAGUE PROMPT

"Add error handling to the service."

Result: Generic try-catch blocks. Wrong patterns.

Wasted tokens. Code you'll rewrite.

PRECISE PROMPT

"Add try-catch to all REST client calls in the resolver layer. Log exceptions with tenant ID. Return default empty response. Follow OrderResolver.java pattern."

Result: Production-ready code. Matches team conventions. Minimal review needed.

AI WRITES THE TESTS. YOU OWN THE QUALITY.

DDD

Domain-Driven Design

Start with the business language. Your code should mirror real concepts, not technical abstractions.

BDD

Behavior-Driven Dev

Define Given/When/Then before writing code. AI generates Gherkin scenarios from your stories.

TDD

Test-Driven Dev

Red → Green → Refactor. AI generates code fast; you provide the quality gate at every step.

100% coverage ≠ quality. AI can hit full coverage with tests that catch zero real bugs.



THE KILLER USE CASE: ONBOARDING

WEEKS → DAYS

New developer ramp-up time

- AI explains any part of the codebase on demand
- Context files give new devs the same knowledge as veterans
- First task plans are generated automatically from Jira tickets
- Reusable prompt libraries encode team best practices

THE NON-NEGOTIABLES

DO

- Always plan before implementing
- Review every line AI generates
- Keep context files current
- Reference existing patterns in prompts
- Save winning prompts for reuse
- Treat AI output like a junior dev's PR

DON'T

- Blindly accept generated code
- Trust AI test results at face value
- Use AI for things you know faster
- Write vague prompts and hope
- Skip planning to "save time"
- Let context files go stale

AI doesn't replace your expertise. It *amplifies* it.

The developers who win aren't the ones with the best tools.
They're the ones who bring clear thinking, structured workflows,
and relentless context to the table.

