The Modern Development Stack

How AI-First Engineers Choose Their Tools

PARADIGM · MODELS · INFRASTRUCTURE

THE FRAMEWORK

Every serious AI workflow has three decisions to get right

01	**Paradigm**	Where do you sit in the loop?
02	**Models**	Which model do you use for which task?
03	**Infrastructure**	What holds it all together when it fails?

Most teams get one right. This talk is about getting all three right.

PART ONE

Paradigm

You're not picking a product. You're picking a relationship with code.

PARADIGM

Where do you want to sit in the loop?

CURSOR	CLAUDE CODE
You are the programmer. AI accelerates you.	You are the decision-maker. AI executes.
You stay inside the editor	You describe outcomes, not steps
Every change is visible inline	Agent reads the whole repo
AI is your co-pilot — you fly the plane	You review diffs, not keystrokes

PARADIGM

Two different relationships to code

CURSOR

Driving a car

You have your hands on the wheel. The car makes you faster. You're still driving.

CLAUDE CODE

Dispatching a driver

You say where you need to go. The driver handles the route. You focus on the destination.

> Neither is wrong. They're different tasks.

PARADIGM

Every new AI coding tool follows the same split

IN THE CODE

Cursor
The original AI IDE

GitHub Copilot
Deep VS Code / GitHub integration

Windsurf
Cascade agent in an IDE

Zed AI
Speed-first, minimalist editor

ABOVE THE CODE

Claude Code
Terminal agent, full repo context

OpenAI Codex CLI
Agentic coding in your terminal

Gemini CLI
Google's terminal coding agent

Devin / CI agents
Headless agents in pipelines

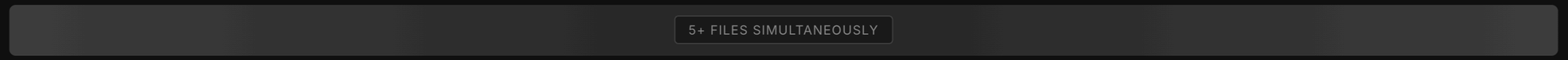
The question for any new tool: does this put me in the code or above it?

PARADIGM

Codebase size changes everything

Cursor territory
Small / greenfield

Claude Code territory
Monorepo / enterprise



Why Cursor degrades at scale

Context management is manual — you're @-mentioning files. The model loses the thread across large surfaces. Tab completion still works, but agent tasks become error-prone.

Why Claude Code holds up

Reads the whole repository before touching anything. CLAUDE.md encodes your architecture decisions. Cross-file reasoning is the default, not an exception.

PARADIGM

Claude Code's real weaknesses

Name these before your audience does.

WEAKNESS	WHAT IT MEANS
Model lock-in	Anthropic only — you can't route cheap tasks to cheaper models.
Cost unpredictability	Heavy workloads run at Opus rates whether the task warrants it.
Spec quality = output quality	CLAUDE.md and your prompt are load-bearing. Vague input produces vague output at scale.
No tab completion	The UX that Cursor perfected — ambient, inline, instant — doesn't exist here.

PARADIGM

Tab completion is still one of the best AI UXs ever built

> *Cursor's tab completion is fast enough to feel predictive, not reactive. The feedback loop is measured in milliseconds — which is why it builds muscle memory.*

Ambient, not prompted	You don't ask for it. It appears. That UX difference matters enormously for flow state.
Inline diff review	Every change is staged in the editor. Accept or reject per hunk. You never lose track of what changed.
Speed wins on simple tasks	For utility functions, small edits, and boilerplate — Cursor completes the developer loop faster.

Neither tool makes you faster by default.

Speed comes from the workflow you build around the tool — not the tool itself.

METR STUDY 2025

19% slower

developers predicted 24% faster

METR STUDY 2026

18% faster

mastery changes the equation

PARADIGM

Claude Code leans into spec-driven development

Spec-Driven Development

- Define what "done" looks like before the agent starts
- High-level intent → agent handles implementation details
- Forces you to think at the architecture level, not the keystroke level

What Goes in CLAUDE.md

- Architecture decisions & module boundaries
- Naming conventions & code style rules
- What NOT to touch (protected files, patterns)
- Testing requirements & definition of done

> CLAUDE.md is a load-bearing file. Spec quality directly determines output quality.

PARADIGM

Finding your paradigm: follow the friction

If you hate losing control of what changes

→ Cursor

If you hate being dragged into implementation details

→ Claude Code

> Don't philosophize about it. Spend $40 for one month on both. You'll know within a week. The friction you feel will tell you more than any comparison article.

The paradigm you choose reflects what you believe your highest-value contribution is —

writing code, or deciding what code gets written.

PART TWO

Models

The developers getting the best results in 2026 are not loyal to one model. They are strategic.

MODELS

We default to one model for everything

- Most teams pick a flagship model and use it end-to-end
- This feels safe — but it's expensive and leaves quality on the table
- The question isn't "which model is best?" — it's "best for what?"

> *"The developers getting the best results in 2026 are not loyal to one model. They are strategic about which model they use for each task."*

MODELS

The coder and the reviewer shouldn't be the same model

CODER

Generate, implement, iterate

Commodity performance at this stage. Mid-tier models match flagship quality on SWE-bench. 5× cheaper.

REVIEWER

Validate, catch blind spots, challenge assumptions

This is where the quality gap is real and measurable. Use your strongest model here.

> Models have characteristic failure modes — patterns they tend to miss. If you review with the same model that coded, you're asking it to catch its own blind spots.

MODELS

Remove context entirely for research

The problem	When a model knows what you're building, it anchors answers to support that direction. It rationalizes bad architectural decisions and overlooks better alternatives.
The fix	A dedicated research/scout agent with zero knowledge of the codebase. It only answers decomposed factual questions — no context, no bias.

> Stripping context isn't a limitation — it's a feature. This applies to any research phase, not just coding.

MODELS

Code generation is a commodity — top models within 1% of each other

SWE-bench Verified — six frontier models within 1.3% of each other

> The harness and scaffolding drive more variance than the model itself. **You don't need Opus-tier pricing for code generation.**

MODELS

Review is where the best model earns its premium

Important comment rate across 25 real PRs with known critical bugs (CodeRabbit benchmark)

> Precision, confidence, surgical fixes — Opus pulls ahead here. **Use your strongest model as the reviewer, not the coder.**

MODELS

Open source is genuinely competitive

Qwen 2.5 Coder 32B	88.4% HumanEval — beats GPT-4's 87.1%
MiniMax M2.5	80.2% SWE-bench (open-weight)
DeepSeek V3.2	MIT license · $0.28/$0.42 per million tokens

> Enterprises routing 80% of requests to open-weight models are seeing **60–70% cost reductions** with no measurable quality degradation. Open-source is the default starting point, not the budget fallback.

MODELS

There is a 25× price gap between cheapest and most expensive frontier model

MODEL	INPUT	OUTPUT
Claude Opus 4.6	$5/M	$25/M
Claude Sonnet 4.6	$3/M	$15/M
Gemini 3.1 Pro	$2/M	$12/M
MiniMax M2.5 (OS)	$0.30/M	$1.20/M
DeepSeek V3.2 (OS)	$0.28/M	$0.42/M

> A well-designed multi-agent system can replace a $200/month flagship subscription with a **$45/month workflow** — at the same or better output quality.

MODELS

Model diversity gives you better quality AND lower cost

STAGE	BEST MODEL TYPE	WHY
Research / Scout	Cheap, fast, no context	Unbiased answers — no anchoring to existing code
Code generation	Mid-tier (Sonnet-class or OS)	Commodity performance, 5× cheaper than flagship
Code review	Strongest available	Where the quality gap is real and measurable

Using different models at different stages catches bugs no single model finds in itself — and routes expensive tokens only where they genuinely matter.

Building a great LLM workflow is less about which model you choose — and more about which model you choose for which task.

The model is one variable. The architecture is the product.

PART THREE

Infrastructure

The model is commodity. The harness is moat.

INFRASTRUCTURE

Two things most people confuse

THE AGENT

The AI doing the work

Model + instructions + tools. Most optimization effort goes here. Usually the wrong place to focus.

THE HARNESS

The environment around the agent

Controls, constraints, context, verification, and recovery. This is where reliability actually lives.

> *Model = CPU. Context window = RAM. Harness = operating system.*

INFRASTRUCTURE

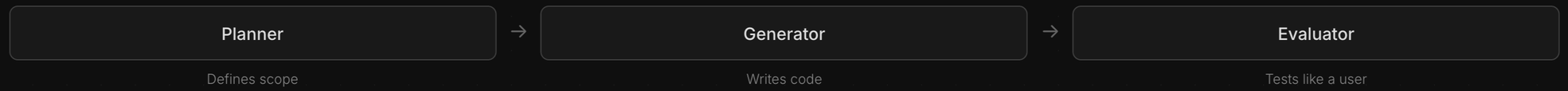
A harness has four jobs

JOB	WHAT IT MEANS
Constrains	What the agent is allowed to do — boundaries, guardrails
Informs	What the agent should know — context, documentation
Verifies	Whether the agent did it correctly — validation, checks
Corrects	What happens when it goes wrong — feedback loops, self-repair

Common mistake: people optimize the agent when the harness is the problem.

INFRASTRUCTURE

Case study: Anthropic's sprint contract pattern



The sprint contract	Generator and Evaluator agree on a definition of "done" before any code is written. Quality is defined upfront, not discovered at the end.
Real-user verification	Evaluator uses Playwright to test like a real user — UI, API, and database. Not just unit tests passing in isolation.

> The harness defined quality, not the model.

INFRASTRUCTURE

Case study: Cognition fixed context anxiety — without changing the model

The symptom	Devin was wrapping up tasks prematurely — rushing to finish before running out of context.
Root cause	As the context window filled, the agent sensed the limit approaching. The model experienced genuine "anxiety" about running out of runway.
The fix	Enabled 1M-token context but capped actual usage at 200K. The model believed it had ample runway — the anxiety vanished.

> No model change. Pure harness engineering. **Some failure modes aren't in the model — they're in the environment.**

INFRASTRUCTURE

The industry got good at watching agents fail — now it needs to prevent it

57%	**Have agents in production**	LangChain 2025, n=1,300+
32%	**Cite quality as #1 production killer**	The most common reason agents are pulled back
89%	**Have implemented observability**	The industry is good at watching failures
62%	**Have step-level tracing**	The infrastructure to observe is there — prevention is still the gap

95% reliability per step × 20 steps

= 36%

end-to-end success rate

Every unreliable step multiplies against every other. Engineers who ship reliable agents spend more time on harness design than model selection.

INFRASTRUCTURE

A practical harness checklist

Limit execution steps	Hardcode a max to prevent infinite loops
Evaluate the trajectory	Did the agent arrive correctly, or did it get lucky?
Write a progress state file	Lets agents resume and lets you inspect internal state (claude-progress.txt)
Use different models per step	Match model capability and cost to the demands of each stage
Mock everything	Never let an agent touch production data during evaluation
Deterministic baselines	Temperature = 0 during testing to reduce run-to-run variance

INFRASTRUCTURE

What actually works at scale

Model routing by role	Different models for orchestrator vs. executor agents. Never default everything to the same model.
Out-of-band watchdog	Context compaction alone isn't enough. When a session crosses a high token watermark, trigger an explicit reset rail — don't let the orchestrator quietly degrade.
Prefer local resources	A mid-run quota failure from a third-party search API can break an entire workflow. Local tools don't have rate limits.
Standardize wrapper scripts	Spawn agent, run tests, deploy — each a single defined wrapper. Wrappers own environment setup, secrets loading, and structured logging. When something fails, rerun the same wrapper with the same args.

THE AI ENGINEERING STACK

Three decisions that compound

01	**Paradigm**	Match your role to your leverage
02	**Models**	Diversity beats loyalty
03	**Infrastructure**	The harness is the product

Lock-in to a single provider means you absorb their quality regressions with no escape hatch. The teams winning in 2026 treat model selection as a per-task decision, not a per-project one.

The model you pick matters less than the system you build around it.

Pick your paradigm. Diversify your models. Invest in the harness.