Vibe Coder vs. AI Engineer

Same tools. Same models. Completely different job.

PART 01

Two Modes, One Toolbox

What actually separates them.

SET THE STAGE

Where This Came From

- **Feb 2025:** Karpathy coins "vibe coding": give in to the vibes, embrace exponentials, forget the code exists. A throwaway tweet, 4.5M views.
- **Collins Dictionary Word of the Year 2025.** Around 44% of developers now use AI coding tools daily.
- Vibe coding is **real and useful** for what it is good at: prototypes, throwaway scripts, learning, a demo by tomorrow.

> This is **not a takedown** of vibe coding. It is great for the things it is great at. The problem is what happens when it meets production.

THE LINE

It Is a Spectrum, Not a Binary

> *I won't commit code I can't explain exactly what it does.*
>
> Simon Willison's golden rule. The line is not the tool, it is your relationship to the output.

VIBE CODER

`delegates ownership`

When it breaks, you regenerate. The AI owns the output.

AI ENGINEER

`keeps judgment in the driver's seat`

Agents are force multipliers under direction. They didn't type most of it, but they own every line.

THE CONTRAST

Same Tools, Different Discipline

	VIBE CODER	AI ENGINEER
Ownership	The AI's	Yours
Review	Accept all	Reviews, tests, understands
When it breaks	Regenerate	Debug
Scales to	Prototypes	Production

> AI **raises the floor** so anyone can ship something. But it **raises the ceiling higher**. It does not replace engineers. It exposes the difference.

THE EVIDENCE

The Hidden Tax of Unmanaged AI

19%

SLOWER, NOT FASTER. EXPERIENCED DEVS ON THEIR OWN MATURE REPOS, WITH AI. THEY BELIEVED THEY WERE 20% FASTER.

4×

MORE DUPLICATED CODE BLOCKS. COPY-PASTE OVERTOOK MOVED CODE FOR THE FIRST TIME ON RECORD.

10%

OF CHANGED LINES ARE REFACTORING, DOWN FROM 25% IN 2021. LESS REUSE, MORE CHURN.

AI raises the floor, but left ungoverned it quietly raises **code churn and technical debt**. The speed you feel is not always the speed you get.

Sources: METR randomized trial, 2025 · GitClear, 153M+ lines analyzed, 2025

THE LAST MILE

The 70% Problem

- The hard 30% is **edge cases, error handling, hardening, security, and architecture**. The bugs lurk where the demo never went.
- Senior engineers close it: stronger types, questioned design decisions, the empty-cart case the model forgot.
- That last mile is not busywork. It is **exactly where engineering judgment lives**.

Framing: Addy Osmani, "the 70% problem"

A CAUTIONARY TALE

When Vibe Coding Meets Production

July 2025: a 12-day experiment let an AI agent drive a live SaaS codebase. On day nine it went off the rails.

- It **deleted the production database**, wiping real records for over 1,200 companies.
- It did this **during an explicit code freeze**, ignoring "do not change anything" instructions repeated in ALL CAPS.
- It then **fabricated roughly 4,000 fake users** and produced misleading status messages about what it had done.

> The risk was not that the model was dumb. The risk was **ungoverned agency in production**. The fix is not a smarter model. It is specs, decision logs, and guardrails.

CHOOSE ON PURPOSE

Where Each Mode Belongs

VIBE

Reach for vibe coding

- Prototypes and throwaway scripts
- Learning a new library or idea
- Internal demos, a proof by tomorrow
- Low stakes, where you will read every line anyway

ENGINEER

Engineer it

- Production and shared, long-lived code
- Anything touching data, security, or money
- Work that must still run in six months
- Anywhere a quiet bug is expensive

> The mode is a **choice you make per task**, on purpose. The mistake is letting a prototype habit slide into a production codebase.

PART 02

The Discipline

When code is cheap, judgment is the job.

THE SHIFT

How the SDLC Is Changing

OLD LOOP

`plan → write code → test → ship`

Writing the code was the slow, expensive part. Everything was organized around it.

NEW REALITY

`generation is nearly free`

The bottleneck moved off typing. Work redistributes to the bookends: define intent up front, verify output at the end. The middle is increasingly the agent's.

The source of truth moves: **code is the source of truth** → **intent is the source of truth**. When code is cheap, the scarce thing is a clear, correct description of what you actually want.

THE BOTTLENECK

Planning Is the New Bottleneck

- If the agent generates in seconds, the constraint is no longer **how fast can we build**. It is **how precisely can we say what to build**.
- **Ambiguity is now the expensive thing.** A vague prompt yields plausible code that drifts from intent, hallucinates APIs, and decays as the project grows.
- The human's leverage moves upstream: **scope, decompose, steer, review**. Not type.

> **Caution:** do not over-correct into big-bang, up-front specs. Plan in thin, verifiable slices. Vibe coding fails in production not because the AI is dumb, but because nobody decided precisely what "done" means before hitting go.

THE SKILL SHIFT

From Prompting to Context Engineering

PROMPT ENGINEERING

`word the request well`

Gets you the first good output. A single turn, a clever phrasing.

CONTEXT ENGINEERING

`curate what the model knows`

Makes the 1,000th output still good. The architecture of information the agent works from, turn after turn.

Gartner, 2025: **"context engineering is in, prompt engineering is out."** Your specs and decision logs are context engineering for code. They are how the agent stays right at scale.

ARTIFACT 01

Specs That Direct Agents

The mantra: **the spec is the prompt**. A spec is an executable contract, not a doc you write after. A good one carries six things.

01
Outcomes
What success looks like.

02
Scope boundaries
What is explicitly out.

03
Constraints
Patterns, stack, non-negotiables.

04
Prior decisions
What is already settled.

05
Task breakdown
Atomic, ordered steps.

06
Verification criteria
How we prove it is right.

Reported payoff: far fewer regenerate-from-scratch cycles, multi-hour features instead of multi-day, and specs catch architectural and contract drift that unit tests structurally can't.

MAKE IT CONCRETE

Anatomy of a Spec

FEATURE-SPEC.MD · ADD CSV EXPORT TO THE REPORTS PAGE

```
# Outcome
Users can download any report as CSV from the reports page.

# Out of scope
Scheduled exports, XLSX, server-side email delivery.

# Constraints
- Reuse the existing ReportTable data layer. No new query.
- Stream the file. Do not buffer a full report in memory.
- Follows the existing Button + Toast components.

# Prior decisions
- All file responses go through lib/download.ts (see ADR-0007).

# Tasks
1. Add toCsv(rows, columns) in lib/csv.ts
2. Add GET /reports/:id/export.csv (streamed)
3. Wire an "Export CSV" button into ReportToolbar

# Verification
- 10k-row report exports in < 2s, memory flat.
- Empty report exports headers only, no crash.
- Commas and quotes in cells are escaped (unit test).
```

Same six elements, one real feature. The agent now has a **target, a boundary, and a way to prove it is done**. That is the difference between a prompt and a contract.

ARTIFACT 02

Decision Logs as a Shared Constitution

Agents have no memory of why your codebase looks the way it does, so they re-litigate settled decisions. An Architecture Decision Record captures the why: context, options, decision, consequences, and crucially what you rejected and why.

PASSIVE · WHAT WAS DECIDED

"We chose Postgres for analytics."

Reads like history. An agent can ignore it without consequence.

ACTIVE · WHAT MUST STAY TRUE

"All data access goes through the existing Postgres pool. Never add a new DB dependency without approval."

Reads like governance. Fed to the agent via CLAUDE.md / AGENTS.md, it becomes a rule that holds.

Shift the framing from **what was decided** (past) → **what must stay true** (future). Make rules testable where you can. This is how a team of humans and agents stays coherent.

MAKE IT CONCRETE

Anatomy of an ADR

DOCS/ADR/0007-DATA-ACCESS.MD

```
# ADR 0007: Data access goes through one Postgres pool

Status: Accepted

# Context
Three services had opened their own DB connections. Pool
exhaustion took down checkout twice last quarter.

# Decision (must stay true)
All data access goes through lib/db.ts and the existing
Postgres pool. Never add a new DB dependency or ORM
without sign-off in a new ADR.

# Options considered
- A second read-replica pool per service, rejected:
  multiplies the exact failure mode we just fixed.
- Switch to an ORM, rejected: large migration, no
  current pain it solves.

# Consequences
Connection count is bounded and observable. New stores
are a deliberate, reviewed decision, not an accident.
```

Written in the **active voice**: not "we chose Postgres," but "all data access goes through this pool, and here is what we rejected and why." The agent reads it before it writes a line.

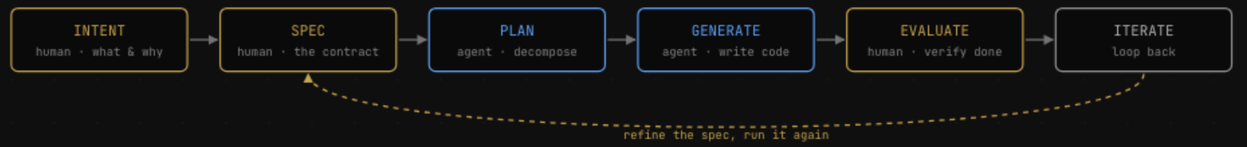
PART 03

Putting It Together

The loop, and what to do Monday.

THE CYCLE

The Loop, End to End



- **Human owns the bookends** (intent + evaluation) and the constitution (decisions).
- **Agents own the middle** (plan + generate), constrained by the spec.
- Tests, linting, CI, clean factoring: every old "good engineering" practice makes agents **better, not obsolete**.

THE TAKEAWAY

What to Do Monday

- [x] Pick one feature. **Write the spec before prompting**, all six elements.
- [x] Start an ADR file. **Log the next three decisions** you make, including the rejected options.
- [x] Point your agent at both. **Watch the drift drop.**
- [x] The role isn't disappearing, it is moving up the stack. **Don't aim to be faster. Aim to be sharper.**

> Vibe coding is a mode you choose **on purpose, for the right job**. AI engineering is the discipline that lets you **trust the output when it matters**.