From Vibe Coding to Agentic Engineering

The Evolution of AI Developer Tools

Kilo Code

Brendan O'Leary

Developer Relations Engineer @ Kilo Code (kilo.ai)
· Advisor to startups

On a mission to help people leverage AI effectively in their workflows. Built a career working with customers as a contractor to US SOCOM, at GitLab, and as a CNCF board member. Every company is a software company — and needs software and AI operational excellence.

Outside work: 1-4 kids hanging off me at any given time, or sneaking away to build something in the workshop.

The Paradigm Shift

From autocomplete to true collaboration

Key Takeaways

Vibe ≠ Agentic

Reject blind acceptance. Keep humans in the loop.

Specialized workflows

Research → Plan → Implement beats one-shot prompts.

Human skills rise

System design, framing, evaluation matter more.

Approval-based control

Productivity improves without losing agency.

Questions to Consider

How will AI-assisted development change the skills most valued on your team?

Where is the right balance between automation and human oversight?

Which parts of your workflow benefit most from specialized AI modes?

How should teams adapt as AI becomes a true collaborator?

Vibe Coding vs. Agentic Engineering

Vibe Coding

Blind acceptance of AI output

Fast demos → fragile systems → debugging debt

Agentic Engineering

Human-led, AI-accelerated

Plan → verify → approve → ship with confidence

The goal isn't more code — it's more **control**

"We are no longer just using machines, we are now working with them."

— Armin Ronacher, creator of Flask

The mental model shift:
AI as an **energetic, enthusiastic junior developer**

The Agent-Model Architecture

Three-way collaboration: You + Agent + Model

You

Direction & judgment

The thinking

Agent

Runs locally in your IDE

Kilo Code, Cursor, Claude Code

Model

Runs in the cloud

Claude, GPT, Gemini

⚠️ Models are **stateless** — every API call starts fresh

The only way to get better output is to put better tokens in

"LLMs are like a new operating system. The LLM is the CPU, the context window is the RAM."

— Andrej Karpathy

Context Engineering

The #1 skill for working with AI

"Context engineering is the delicate art and science of filling the context window with just the right information for the next step."

— Andrej Karpathy

Cognition (Devin creators):
"Context engineering is effectively the **#1 job** of engineers with AI"

The Four Strategies

✍️ Write Context

Save information outside the context window

`Scratchpads, memories, plan files`

🎯 Select Context

Pull the right information into the window

`RAG, file references, @-mentions`

🗜️ Compress Context

Retain only tokens required for the task

`Summarization, trimming`

🔀 Isolate Context

Split context across sub-agents or sandboxes

`Parallel agents, task separation`

The Cost Reality

Context scales quadratically — twice the context requires 4x computational power

10K	200K	600K	1M+
tokens	tokens	tokens	tokens
~$0.03	~$0.60	$1.50+	💸💸💸

More context ≠ better results

You're paying for noise, not signal

Context Poisoning

Once mistakes enter context, they contaminate everything

☠️ Sources of Poison

Task mixing
Unrelated context from previous tasks

Outdated comments
Stale documentation and TODOs

Massive debug logs
Thousands of irrelevant lines

Hallucinations
AI mistakes that compound

✅ The Only Reliable Fix

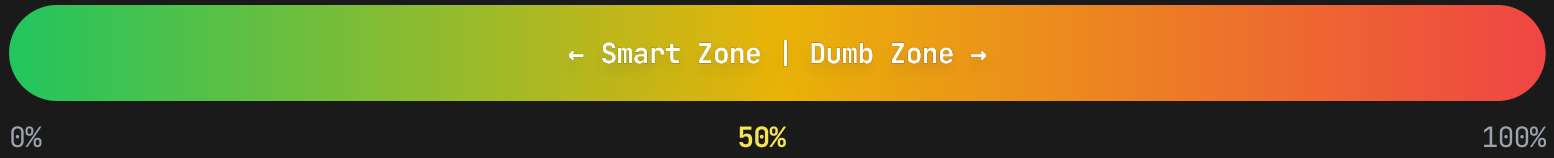
Start a new task

Clean context = clean thinking

Don't try to "fix" poisoned context — just start fresh

The "Dumb Zone"

Around 40% context utilization, you start seeing diminishing returns



⚠️ The MCP Trap

Too many MCP servers = working entirely in the dumb zone

Each server adds to your system prompt

✅ The Rule

Stay under 50% context utilization when possible

— Dex Horthy

Research → Plan → Implement

The workflow that went viral on HackerNews

The Classic Mistake

❌

What most people do

"Hey AI, implement this feature"

Jump straight to code → wrong assumptions → wasted time → frustration

✅

What works

Research → Plan → Implement

Understand first → explicit steps → execute with confidence

"A bad line of research = potentially hundreds of bad lines of code"

— Dex Horthy

Phase 1: Research

🔍 What You Do

Understand the system
How does this actually work?

Find the right files
Where does this code live?

Stay objective
No implementation yet — just learning

📄 Output

Research Document

- Exact files involved
- Line numbers for key functions
- How data flows through the system
- Dependencies and side effects
- Edge cases to consider

Use **Ask Mode** — prevents overeager changes

Phase 2: Plan

📝 What You Do

Outline exact steps
File names, function names, code snippets

Include testing
How to verify after every change

Be explicit
What will change, what won't

📄 Output

Plan File

- Step-by-step instructions
- Specific code changes
- Test commands for each step
- Rollback strategy

> So clear that even a "dumb model" couldn't screw it up

Use **Architect Mode** — planning and design focus

Phase 3: Implement

⚡ What You Do

Execute the plan
Step by step, no improvising

Keep context low
Fresh task for each major step

Review each change
Before moving to the next step

Commit frequently
Small, reversible changes

🎯 The Key Insight

Human review at **research** and **planning** stages is the highest leverage use of your time

By the time you're implementing, the hard thinking is done

Use **Code Mode** — file editing and implementation

"AI cannot replace thinking. It can only amplify the thinking you have done — or the lack of thinking you have done."

— Dex Horthy

Modes and Customization

Same model, different instructions — modes define the AI's role

❓

Ask Mode

Questions & exploration

Prevents overeager changes

💻

Code Mode

File editing & implementation

The workhorse

🏗️

Architect Mode

Planning & design

Big picture thinking

🌍 Global Rules

Apply to all projects

📁 Workspace Rules

Project-specific context

🎭 Mode Rules

Behavior per mode

Top Settings & Configuration

Settings that can make a difference in your Kilo experience

🤖 Agent Behavior

- Modes configuration
- MCP servers
- Rules & workflows
- Skills

⚙️ Auto-Improve

- Check-in frequency
- Read/write permissions
- Execution controls

✨ Autocomplete

- Chat autocomplete
- Background editing (experimental)

💡 Start with defaults, then tune as you learn what works for your workflow

Mental Model for Agent Configuration

Where does configuration belong in your repo?

📋 AGENTS.md

Always-on rules for the repo

`"README for AI agents"`

- Project conventions
- Build commands
- Testing requirements

🎯 SKILLS.md

Task-specific workflows (on-demand)

`Reusable playbooks`

- "api-design"
- "code-review"
- .kilocode/skills/

🎭 Modes

Personas with different behaviors

`Role-based configurations`

- Architect
- Coder
- Debugger

📢 Memory Bank is being deprecated in favor of AGENTS.md + SKILLS.md

Kilo Code Power User Tips

Get more out of every interaction with these features

📎 @ Mentions for Context

@file — Pull specific files into context
@commit / @URL — Reference commits or web pages
@terminal — Include terminal output directly

⚡ Slash Commands

/mode — Switch between Ask, Code, Architect
/newtask — Start fresh with clean context
/condense — Compress context when it gets long

💡 Code Selection

Highlight code → right-click to add to context

✨ Autocomplete

Works in prompts and code

Beyond the IDE: CLI, Cloud Agents, Slack Bot

Use Kilo Code wherever you work — terminal, browser, or team chat

Model Context Protocol (MCP)

The fundamental problem: AI models only receive text and return text

🔌 What MCP Does

- Open-source standard from Anthropic
- Connects AI to external resources
- One server works with all supporting agents
- Read files, run commands, call APIs

📦 Examples

GitHub MCP
67 tools for issues, PRs, workflows

Context7
Up-to-date framework documentation

Amazon
Thousands of internal MCP servers

⚠️ Each MCP server adds to your system prompt

If you don't use it, disable it — this is context engineering applied to tooling

Working with Internal Platform APIs

How to give Kilo context about platform APIs you're building against

1️⃣ Swagger / OpenAPI Spec

Pull it in directly via @file mention

Best for: Standard REST APIs

2️⃣ Convert to Markdown

Document in AGENTS.md for always-on access

Best for: Stable internal APIs

3️⃣ Reference URL/URI

Point to hosted docs that update frequently

Best for: Rapidly changing APIs

4️⃣ Build an MCP Server

For complex, multi-step interactions

Best for: Multi-project platforms

Security & Professional Responsibility

25-33% of AI-generated code contains potential security vulnerabilities

AI learned from tutorials, prototypes, and "I'll fix this later" code

- 🔐 Never paste secrets, API keys, or customer data
- 🔍 Run security scanners before merge
- 📝 Explicitly specify security requirements
- 📋 Know your AI provider's data policies
- ⚠️ Extra scrutiny for auth & user data
- 🚫 Never assume AI knows current best practices

"You can outsource the code writing, but you can't outsource the security responsibility"

Working Alongside Kilo

How do I avoid getting my manual edits overwritten?

⚠️ The Problem

Editing the same files Kilo is working on leads to conflicts

✅ The Solution

Use CLI with parallel mode (git worktrees)

📁
You work here
Main directory

↔️
Isolated
No conflicts

🤖
Kilo works here
.kilocode/worktrees/

Review Kilo's work as a **separate PR**

Roadmap

Expect improvements to the key features you use getting richer and deeper

🌐 Expanded Surfaces

VS Code

JetBrains
Re-architected extension

CLI
New release

App Builder & Cloud Agents

🎯 Current Focus

🔧 **Feature Depth**
Building out richness of each feature

🔗 **Interconnectedness**
Better integration between features

✨ **Agent Experience**
Consistent quality across all surfaces

The Future is Now

😊

"I'm having more fun programming than I ever had in 52 years."

— Kent Beck

"LLMs will change software development to a similar degree as the change from assembler to high-level languages."

— Martin Fowler

~50%
developers use AI weekly

~4 hrs
median time saved per week

10x
more code (Google preparing)

Call to Action

🎯

Pick one tool and get reps

Mastery comes from practice, not theory

📋

Try Research → Plan → Implement

On your next feature or bug fix

🧠

Master context engineering

It's the highest leverage skill

💭

Don't outsource the thinking

AI amplifies — it doesn't replace

"The whole landscape of what's 'cheap' and what's 'expensive' has shifted. Things we didn't do because we assumed they were expensive or hard just got ridiculously cheap. So, we just have to be trying stuff!"

— Kent Beck

Thank You

Questions?

Start experimenting today