

The Era of "Vibe Coding" is Over.

It's Time to Ship Production Code

Who am I?

A little bit about me..

- **Product Manager, LawDepot**
- IT & Business System Lead, Edmonton Unlimited.
- Tech consulting — Advising big firm on tech solution & system architecture.
- Built my own startup; Worked for startups.
- *All this to say, I'm well versed in art of "**shipping production code**"*



Kundan Deep (KD) Singh
Product Manager, LawDepot

kdsingh.ca
connect@kdsingh.ca
[LinkedIn](#)



Agenda

What are we going to cover today?

Theory

01. Founder PM Era
02. Evolution of AI Coding
03. 8 Phases of AI Development
04. Live Build

Tools

- Any AI model
(Claude, ChatGpt, Gemini)
- Github CLI
- Any IDE or Agent Manager
(ClaudeCode, Codex, Antigravity)

Housekeeping

What to keep in mind as we go through the workshop

- Live demos are always a challenge.
- Let's work together as we navigate any bugs, glitches, or unexpected surprises that may pop up during the build.
- I've simplified some of the concepts to make them more accessible and easier to follow.
- Focus on the core functionality of the app.
- Rapid prototyping to test your ideas — not worried about state of art security, auth, or 2fa/mfa etc.
- Don't worry about taking detailed notes.

Founder PM Era

The great shift in responsibilities

The AI as a Senior Engineer; The Founder as a Technical Product Manager

- Your job is to define the **"What"** and not the **"How"**
- **"How"**
Technical Founders spend 80% of time wrestling with syntax, dependencies, and architecture; which AI already knows best.
- **"What"**
What AI doesn't know is your business logic, your unstated assumptions, your customers, your vision of the product.
- When code generation is free and instant, the bottleneck moves from technical ability to product taste, domain mastery, and system guardrails

Evolution of AI Coding

5 Eras of AI coding

1. Inline / Autocomplete Coding (2021–2023)
2. Conversational / Chat-Driven Development (2023–2024)
3. Vibe Coding / Rapid Prototyping (Mid 2024–Present)
4. Spec-Driven / Context-First Development (2024–Present)
5. Agentic / Autonomous Workflows (2025–Present)

Inline / Autocomplete Coding (2021–2023)

1st Era of AI coding

The Intelligent Typewriter

- Early Copilot predicting next lines/functions in-flow.
- Zero architectural awareness. It helps you type boilerplate faster, but it has no idea what your database looks like, what design patterns your team uses, or what files exist outside the current tab. You still need to know how to code.

```
JS test.js 1 ●
JS test.js > calculateDaysBetweenDates
1 function calculateDaysBetweenDates(begin, end) {
    var beginDate = new Date(begin);
    var endDate = new Date(end);
    var days = Math.round((endDate - beginDate) / (1000 * 60 * 60 * 24));
    return days;
}
2
```

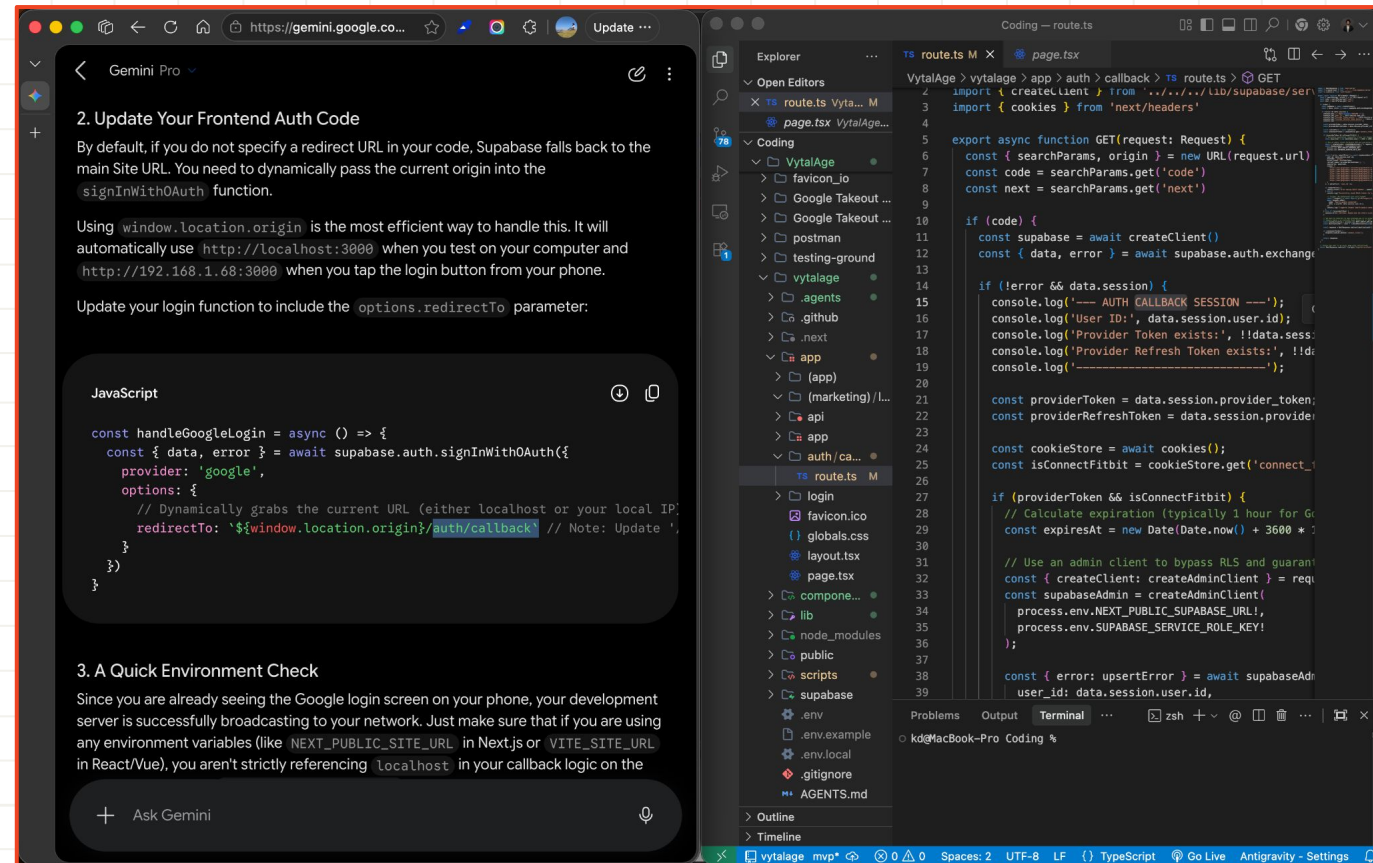
Conversational / Chat-Driven Development (2023–2024)

2nd Era of AI coding

The StackOverflow on Steroids

- Copy-pasting errors and snippets into AI chat/web interfaces.
- Severe context window exhaustion & isolated fixes. The AI fixes the bug in a vacuum without seeing the rest of your app.

Copy-pasting isolated snippets leads to conflicting styles, duplicated helper functions, and brittle spaghetti code.



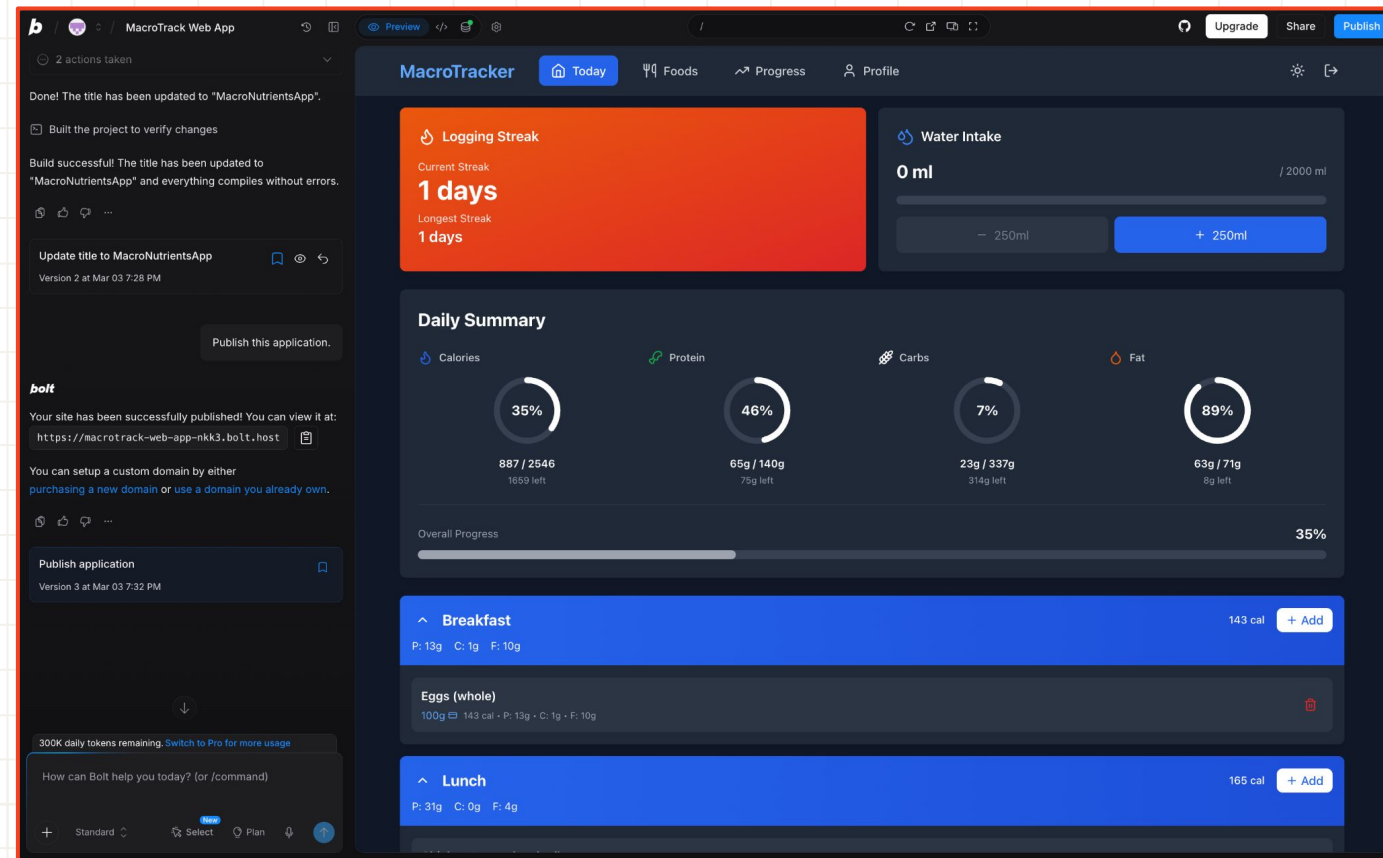
Vibe Coding / Rapid Prototyping (Mid 2024–Present)

3rd Era of AI coding

The Zero-to-One

- Loose natural-language prompts in Bolt.new, Lovable.dev, Claude Code, Codex
- Great for prototypes, disastrous for production. Adding feature #10 breaks feature #1. Without a formal data model and boundaries, the AI collapses under its own weight.

The most dangerous era since it's hardest to break away from.

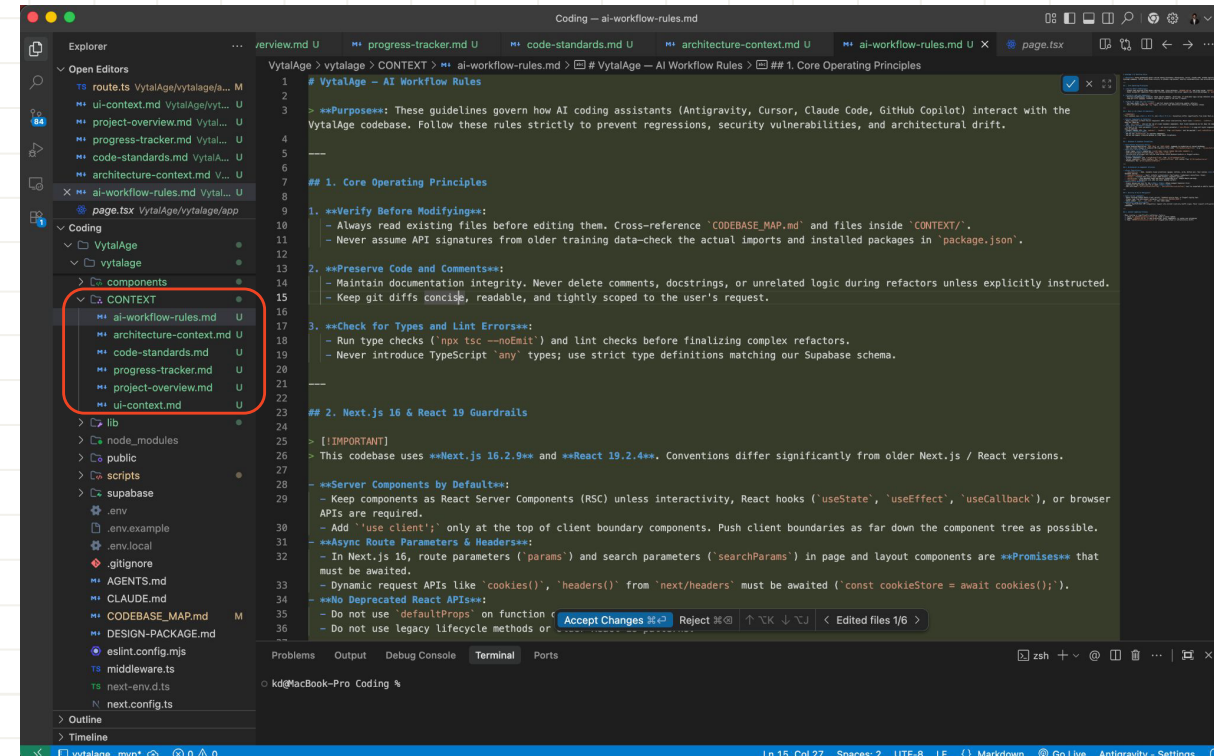


Spec-Driven / Context-First Development (2024–2025)

4th Era of AI coding

The Senior Engineer

- Writing specs, data models, API contracts, RULES.md, and CONTEXT.md first; constraining AI execution.
- High architectural integrity and maintainability; predictable code generation.
- Front-load the thinking. Founder acts as the Systems Architect or as Senior Engineer defining constraints, variables and ubiquitous language; the AI acts as a constrained junior implementer that cannot stray off course.



Agentic / Autonomous Workflows (2026–Present)

5th Era of AI coding

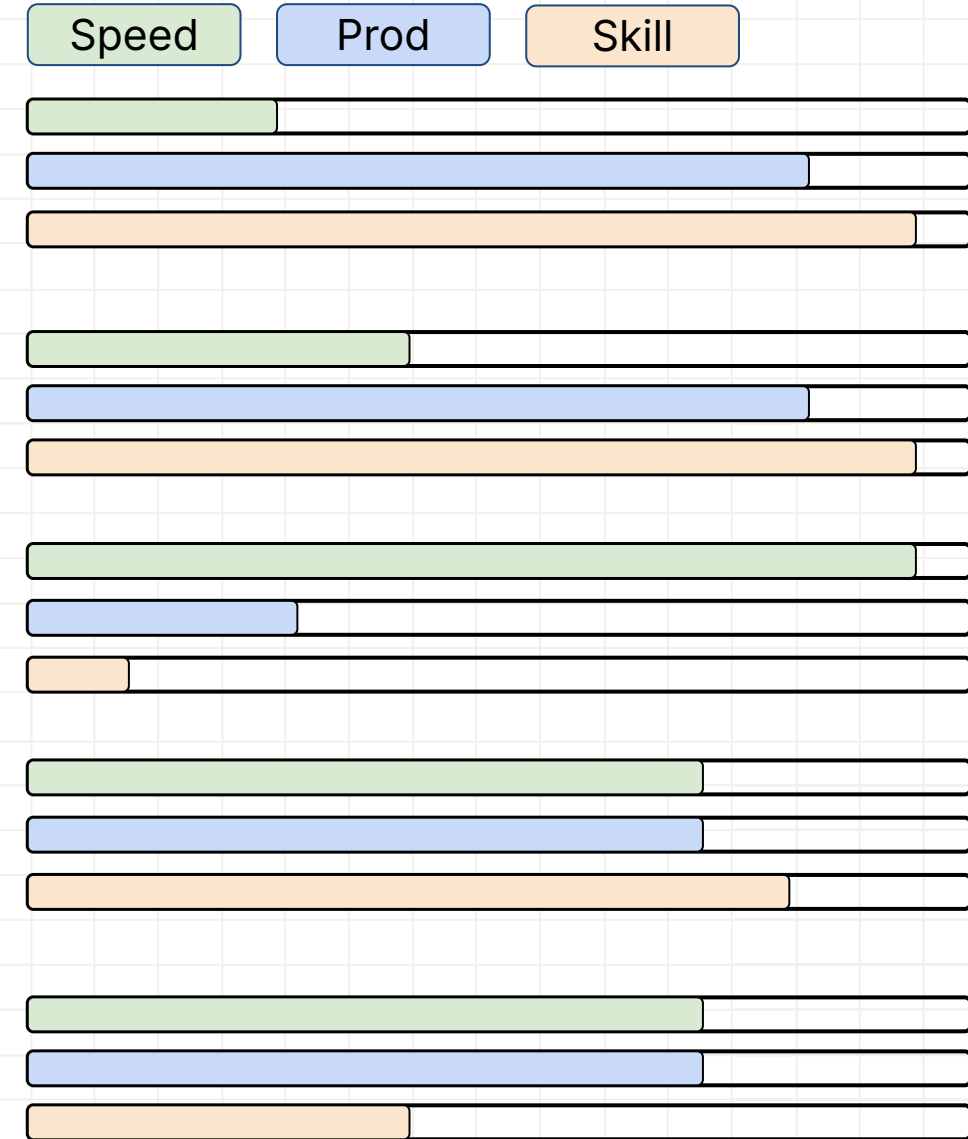
The AI as a Senior Engineer; The Founder as a PM

- The AI acts as the lead architect and builder. It actively interviews the founder to extract domain nuances, user requirements, and edge cases, autonomously translating business intent into system architecture, specs, and implementation plans.
- The founder steers high-level product intent and user outcomes rather than micromanaging code and schemas.
- Autonomous agents break down goals, write code, run tests, and self-correct using AI development phases, structured workflows and domain skills.

Constraints

Speed vs. Production Quality vs. Skill needed

1. Inline / Autocomplete Coding
(Skill constrained)
2. Conversational / Chat-Driven Development
(Skill constrained)
3. Vibe Coding / Rapid Prototyping
(Prod Quality constrained)
4. Spec-Driven / Context-First Development
(Skill constrained)
5. Agentic / Autonomous Workflows
(Overall best optimized)
(Speed can be improved even further — Software Factories)



Speed higher is better || Prod Quality higher is better || Skill lower is better

8 Phases of AI Development

AI Development lifecycle

1. Idea / Feedback

Capturing raw product inspiration, user pain points, or stakeholder requests that trigger the need to build something new.

2. Spike / Research (optional)

A time-boxed exploration to gain knowledge, test assumptions, and de-risk unknown technical factors.

3. Prototype (optional)

Converting functional requirements into "concept slice" to explore UI, workflows, and product taste through rapid iteration.

4. PRD/Specs

Converting concept slice into rigorous operational specs; definitive commitment to system's end state.

5. Implementation Planning

Deconstructing the PRD/Spec into atomic user stories, dependency maps, and phased execution seams before writing a single line of code.

6. Execution

Deploying autonomous coding agents to build user stories in continuous loops, enabling parallelized away-from-keyboard (AFK) development.

7. QA

Systematically verifying the built system against PRD/Specs through automated testing and human testing.

8. Release

Safely packaging, deploying, and operationalizing the finished software into production for real users.

Framework / Mental Modal

How are we using AI to code in 5th Era

01

Smart Zone vs Dumb Zone

Smart Zone context window is 100k
Token

02

Specific Skills

Matt Pocock's AI Skill

02

General Software Dev Concepts

Such as User stories, Tracer Bullets,
TDD etc

Idea / Feedback

Capturing pain and inspiration

Capturing raw product inspiration, user pain points, or stakeholder requests that trigger the need to build something new.

- Starts with a new app idea, a missing feature request from a customer, or feedback from stakeholders.
- At this point, there is no technical definition, no UI, and no architecture—just a raw signal or problem statement that needs solving (i.e. Function requirements).
- The goal is to refine your idea, articulate the core user value and clarify whether this idea is worth exploring before dedicating resources to it.

/grill-me

/grill-with-docs

Spike / Research

De-risking unknowns

A time-boxed exploration to gain knowledge, test assumptions, and de-risk unknown technical factors.

- Dedicated time to research unfamiliar dependencies such as external APIs, assess library tradeoffs, or evaluate system feasibility when the team lacks clear answers.
- Replaces guesswork with hands-on technical investigation, establishing what is possible, what is costly, and what constraints exist.
- Strictly time-bound to prevent analysis paralysis—producing clear findings, proof-of-concepts, and realistic effort estimates.

/research

Prototype

Validating taste and flow

Converting functional requirements into a tangible "concept slice" to explore UI, workflows, and product taste through rapid iteration.

- Functional requirements describe what is needed but have no UI, layout, or synergy; prototyping turns them into an interactive concept slice.
- This is where you explore user flow, visual hierarchy, and interaction design—defining how the app feels and behaves.
- Rapidly generating multiple throwaway versions to experiment with different directions without worrying about production-grade code.

/prototype

PRD/Specs

Defining the end state

Converting the validated concept slice into rigorous operational specs—the definitive commitment to the system's end state.

- Documents exactly what the system will look like, feel like, and how it will behave under all conditions (data models, state changes, edge cases, error handling).
- Moving from loose exploration and throwaways to a formal, documented contract that the team and AI commit to building.
- The Single Source of Truth: Establishes rigid acceptance criteria, API contracts, and rules (RULES.md, CONTEXT.md) that prevent ambiguity during development.

/to-spec

Implementation Planning

Mapping dependencies and seams

Deconstructing the PRD/Spec into atomic user stories, dependency maps, and phased execution seams before writing a single line of code.

- Break down the large prd/spec into concrete, manageable work items (stories/tickets) that roll up into a clear, phased implementation roadmap.
- Map out critical path relationships—identifying what must be built first to avoid circular dependencies, regressions, and costly rewrites.
- Establish explicit completion and verification criteria for each ticket so the AI engineer has deterministic boundaries and never drifts off course.

/to-tickets

Execution

Autonomous agentic code generation

Deploying autonomous coding agents to build user stories in continuous loops, enabling parallelized away-from-keyboard (AFK) development.

- Coding agents pick up the prioritized tickets and execute against the plan, translating user stories directly into clean, production-grade code.
- Agents iterate in tight feedback loops—writing code, running linters, resolving type errors, and verifying builds before proceeding. A clean roadmap unlocks parallel execution, allowing multiple agents to tackle unblocked tickets simultaneously.
- With proper setup (Spike → Prototype → Specs → Plan), the execution phase runs reliably in the background without needing constant human intervention.

/implement

/implement-spec

QA

Verifying against the spec

Systematically verifying the built system against PRD/Specs through automated testing and human testing.

- Rigorously testing that the actual system behavior matches what was promised in the PRD (business logic, edge cases, failure states).
- Dual-Layered Verification: Combines automated agent sweeps (unit tests, integration suites, lint audits) with human PM acceptance testing.
- Zero Regression Tolerance: Ensuring that adding the new feature didn't introduce hidden side-effects or break existing functionality across the app.

/code-review

Ask agent to create a QA plan for humans

Release

Shipping into live production

Safely packaging, deploying, and operationalizing the finished software into production for real users.

- Automating build pipelines, applying database migrations, and configuring environment variables for a clean rollout.
- Generating user-facing changelogs, updating system architecture docs, and provisioning monitoring/error logging.
- Releasing the feature to users to begin collecting new feedback, seamlessly feeding back into Phase 1 (Idea / Feedback).

“

Thank You

The Era of "**Vibe Coding**" is Over. It's Time to Ship **Production Code**