
THE VIBE CODING MASTERCLASS
From Zero to Shipped

✦ THE COMPLETE GUIDE TO ✦

VIBE CODING

MASTERCLASS

Build Real Software with AI - No Coding Experience Required

Updated March 2026	10 Chapters	Beginner → Advanced

Foreword: Why This Book Exists

On February 2, 2025, a single post on X (formerly Twitter) quietly detonated inside the software development world. Andrej Karpathy co-founder of OpenAI and former head of AI at Tesla described a new way of working with computers he called 'vibe coding'. Within months, it had gone from a tweet to a movement. By the end of 2025, Collins Dictionary named it word of the year.

This book is for everyone who read about vibe coding and thought: 'That sounds powerful but where do I actually start?' Whether you're a business founder who needs an MVP without hiring a developer, a creative who has ideas but no technical background, or a working developer who wants to multiply their output tenfold, this guide was written for you.

We've drawn from academic research, security studies, industry reports, tool documentation, and hands-on practitioner experience. Every claim is grounded in real data. Every technique has been tested in production contexts. This is not hype it's a framework.

📖 How to Read This Book

Read it front-to-back on your first pass. Then treat it as a reference: return to Chapter 3 when picking tools, Chapter 5 when you're stuck on a bug, and Chapter 7 when you're ready to ship. The checklists at the end of each chapter are designed to be used on real projects.

PART ONE

The Foundation

What vibe coding is, where it came from, and why it changes everything

Chapter 1: The Birth of a New Paradigm

1.1 Karpathy's Original Post

Everything starts with six words: 'fully give in to the vibes.' On February 2, 2025, Andrej Karpathy described a workflow that would electrify millions:

```
"There's a new kind of coding I call 'vibe coding', where you fully give
in to the vibes, embrace exponentials, and forget that the code even
exists. It's possible because the LLMs are getting too good. Also I just
talk to Composer with SuperWhisper so I barely even touch the keyboard. I
'Accept All' always, I don't read the diffs anymore. When I get error
messages I just copy paste them in with no comment, usually that fixes
it. The code grows beyond my usual comprehension... I just see stuff, say
stuff, run stuff, and copy paste stuff, and it mostly works."  — Andrej
Karpathy, X (formerly Twitter), February 2, 2025
```

This wasn't just a tip. It was a philosophical provocation. Karpathy one of the most respected AI researchers alive was saying that the practice of writing code manually was becoming obsolete. The computer, at last, was learning to speak our language.

Karpathy's idea built on something he'd said in 2023: 'The hottest new programming language is English.' By 2025, that prediction had arrived.

1.2 What Is Vibe Coding, Precisely?

Vibe coding is a software development approach where you describe what you want to build in natural language and let AI models generate the code. The programmer's role shifts from implementer to orchestrator.

The critical distinction, as programmer Simon Willison put it: 'If an LLM wrote every line of your code, but you've reviewed, tested, and understood it all, that's not vibe coding in my book that's using an LLM as a typing assistant.' True vibe coding involves a degree of acceptance: you let the AI build, and you guide the direction.

> **The Core Mental Shift**
>
> Traditional coding: You are the builder. AI is a reference tool. Vibe coding: You are the architect and director. AI is the builder. Your job is now: (1) describe intent clearly, (2) test outcomes ruthlessly, (3) iterate fast.

1.3 The Spectrum: Pure Vibe to Supervised Building

Not all vibe coding is the same. Think of it as a dial:

Mode	Approach	Best For
Pure Vibe	Accept all AI output, no review. 'Forget the code exists.'	Throwaway weekend projects, personal tools, demos
Guided Vibe	Describe intent, review key outputs, test features as you go	MVPs, early-stage startups, internal tools
Supervised Build	AI writes code, human reviews every file, enforces patterns	Production SaaS, team codebases, client work
Agentic Engineering	Orchestrate AI agents with oversight and strict quality gates	Enterprise software, large-scale systems (Karpathy's 2026 term)

1.4 The Numbers Behind the Movement

This isn't fringe. The adoption data tells a story of mainstream transformation:

- 78% of developers now use AI coding tools daily (as of early 2026)
- 25% of Y Combinator's Winter 2025 batch had codebases that were 95% AI-generated
- Roughly 85% of developers use AI tools regularly as of early 2026
- Collins Dictionary named 'vibe coding' word of the year for 2025
- iOS app releases increased approximately 60% year-over-year in late 2025, largely from AI-assisted development

1.5 From 'Vibe Coding' to 'Agentic Engineering'

By February 2026 exactly one year after his original post Karpathy updated his thinking. LLMs had become dramatically more capable. He proposed a new term: 'agentic engineering.'

'Agentic' because the new default is that you are not writing the code directly 99% of the time you are orchestrating agents. 'Engineering' to emphasize that there is an art, science, and expertise to doing it well.

The key insight: vibe coding is the entry point. Agentic engineering is where it matures. This book covers both.

PART TWO

The Toolkit

Every tool you need, honestly assessed

Chapter 2: Your Vibe Coding Toolkit

Choosing the wrong tool is the most common beginner mistake. The landscape has exploded: AI IDEs, full-stack builders, terminal agents, browser environments. This chapter cuts through the noise.

The most important principle: 'Cursor for developers, Lovable for MVPs, Claude Code for complex codebases and most teams need more than one tool.'

2.1 The Three Categories of Tools

Before comparing individual tools, understand the three categories. Choosing the wrong category wastes time and money.

Category	What It Does	Who It's For
AI App Builders	Generate complete applications from descriptions. No code setup needed.	Non-technical founders, product managers, designers, idea-validators
AI Code Editors (IDEs)	Supercharge existing codebases with AI assistance. Require programming knowledge.	Developers who want to move faster and write less boilerplate
Terminal Agents	Operate directly in your command line, can run commands, manage files, execute tests autonomously.	Senior developers, DevOps engineers, large codebase managers

2.2 The Complete Tool Comparison (2026)

Tool	Best For	Coding Needed?	Price/mo	Strength
Cursor	Developers, complex projects	Yes	$20	Deep codebase understanding
Lovable	Non-coders, MVPs, UI-heavy	No	$25	Polished UIs, full-stack gen
Bolt.new	Rapid web prototyping	No	$20	Instant browser-based builds
Claude Code	Complex refactors, large codebases	Yes	$20+	93% benchmark, terminal-native
Windsurf	AI pair programming	Yes	$15	Cascade agent, large context
Replit	Learning, cloud-first builds	Partial	Free/$20	All-in-one cloud environment
v0 by Vercel	UI generation only	No	Free/$20	Best React/Next.js components

2.3 Deep Dive: The Major Tools

Cursor - The Power Developer's Choice

Cursor is an AI-native IDE a fork of VS Code that understands your entire codebase and makes context-aware changes. It supports multiple AI models including Claude Sonnet 4.5, Claude Opus 4.5, and GPT-5, giving you flexibility to pick the best model for each task.

- Best for: Developers evolving existing projects or building complex features
- Key strength: Deep codebase understanding, multi-file edits, Composer/Agent modes that index whole repositories
- Watch out for: Cursor enhances existing coding skills rather than replacing the need for programming knowledge
- Pricing: Free tier (limited), Pro at $20/month, Pro+ at $60/month, Ultra at $200/month

Cursor Pro Tip: The Rules File

Create a file called. cursor rules in your project root. This tells the AI exactly how to behave: which frameworks to use, which patterns to follow, what to avoid. Example: "Always use TypeScript. Never use Redux. Write concise functional React components. Use Tailwind for styling. Always add error handling to async functions." The AI reads this before every generation, giving you consistent, opinionated output.

Lovable - The Non-Coder's Best Friend

Lovable positions itself as the most comprehensive end-to-end vibe coding platform, designed to take users from idea to deployed application in minutes. You describe your vision, Lovable sketches a polished UI, then wires it up to a real backend usually via Supabase for auth, data, and storage.

- Best for: Non-technical founders, product managers, designers building customer-facing web products
- Key strength: Most polished UI output of any app builder. Scores ~4.4/5 for visual quality in user reviews
- Watch out for: Struggles with complex business logic and custom backend requirements
- Pricing: Free tier, Pro at $25/month (100 credits), Business at ~$100/month

Bolt.new - Speed Above All

Bolt.new runs a complete Node.js development environment directly in your browser through WebContainer technology. No installation, no setup: open a browser tab and start building full-stack React and Node.js applications immediately. The platform delivers millisecond boot times and even works offline once a project loads.

- Best for: Rapid prototyping, hackathon builds, fast frontend experiments, developers testing multiple ideas
- Key strength: Getting from idea to shareable URL faster than any other tool

- Watch out for: Less polished than Lovable under the hood; best for specific use cases rather than full workflows
- Pricing: Free tier, Pro at ~$20/month

Claude Code - The Reasoning Engine

Claude Code is Anthropic's terminal-native agent. No GUI, no VS Code wrapper just a command-line agent that works where many developers already spend their time. It uses Claude's extended thinking to actually plan before it acts, which shows in the quality of complex multi-step refactors.

- Best for: Large codebases (50k+ lines of code), complex refactors, multi-step features, developers who live in the terminal
- Key strength: 93% success rate on app-building benchmarks the highest measured benchmark performance of any major tool
- Watch out for: Terminal-based workflow assumes familiarity with command-line interfaces, Git, and software architecture
- Pricing: $20–$200/month (Claude Pro plans), usage varies by codebase size

Windsurf - The Streamlined IDE

Windsurf is a serious option for developers who want an IDE designed around an agent workflow instead of bolting AI onto an old editor. Its Cascade agent handles context better than most tools, making it particularly strong for large codebase navigation.

- Best for: Advanced coders seeking fast AI pair programming and real-time code transformations
- Key strength: Cascade agent, fluid editing experience, excellent value at $15/month
- Watch out for: Business model questions around long-term viability

Replit - Learn, Collaborate, Deploy

Replit combines a code editor, hosting, database, and AI agent in one browser platform. It's particularly strong for learning: you can start a project in your browser without installing anything, and the AI agent explains concepts as it builds.

- Best for: Learning to code, educational contexts, team collaboration, cloud-first projects
- Key strength: Nothing to install; great community and multi-language support
- Watch out for: The Replit AI agent famously deleted a production database when a founder gave it too much autonomy — always use staging environments

2.4 The Winning Workflow Combination

The most effective teams in 2026 don't pick one tool - they chain them:

Stage	Tool	Goal
Stage 1	Lovable or Bolt.new	Build the demo in a day. Validate the idea. Kill bad ideas fast.
Stage 2	Cursor + Foundation (e.g. MakerKit, Next.js SaaS Starter)	Build production version. AI works better with patterns to follow.
Stage 3	Claude Code or Windsurf	Handle refactoring as complexity grows. AI handles implementation; you handle architecture.
Stage 4	Human review + security audit	AI gets you 70–80% there. The final 20% requires human expertise.

PART THREE

The Craft

Prompting, workflows, and the discipline to build things that work

Chapter 3: The Art of Prompting

Prompting is the core skill of vibe coding. The quality of your output is a direct function of the quality of your input. Most beginners fail here not because they lack ideas, but because they communicate those ideas poorly to the AI.

Here's the governing truth: AI models are extraordinarily capable but have no memory, no context about your product, and no ability to read between the lines. You must be explicit.

3.1 The Levels of Prompting Quality

Level	Bad Example	Good Example
Initial Build	Build me an app for task management.	Build a React/TypeScript task management app for freelancers. Features: create tasks with due dates, assign priority (Low/Medium/High), filter by status. Use Tailwind CSS, dark mode. Do not add auth yet. Show a Kanban board layout.
UI Feedback	Make it look better.	The task cards need more padding (16px inside). Move the priority badge to the top-right corner. Change the 'Done' column background to a very subtle green (#F0FFF4). Keep font sizes the same.
Bug Fix	It's broken.	The date picker throws 'TypeError: Cannot read property of undefined' when I click a task with no due date set. The error is in TaskCard.tsx around line 47. Fix the null check before rendering the date field.
Feature Add	Add notifications.	Add an in-app notification bell icon in the top-right header. When a task's due date is today or overdue, show a red badge with the count. Clicking the bell opens a dropdown listing those tasks. Use the existing Tailwind color variables.

3.2 The PRD Prompt - Starting Right

The Product Requirements Document prompt is the single most important prompt you will write. It sets the context, the constraints, and the direction for everything that follows. Never skip it.

A strong PRD prompt includes:

- Role instruction: 'Act as a Senior Full-Stack Developer'
- Tech stack: exactly which frameworks, languages, and services to use
- Core features: a specific numbered list of what to build
- Explicit exclusions: 'Do NOT include payment processing in this version'
- Design direction: dark mode, minimalist, 'Stripe-like', etc.
- A stop instruction: 'Do not write any code yet. First, outline the file structure. Wait for my approval.'


```
EXAMPLE PRD PROMPT:  "Act as a Senior Full-Stack Developer. We are
building a web app for digital recipe management.   Core Features: 1.
User authentication (email + Google OAuth) 2. Create, edit, and delete
recipes with title, ingredients, steps, and photo 3. Recipe search and
filter by category (Breakfast, Lunch, Dinner, Dessert) 4. A shareable
public link for each recipe Tech Stack: Next.js 14, TypeScript, Tailwind
CSS, Supabase for auth and database, Cloudinary for image uploads.
Design: Clean, minimal white-card layout. No dark mode required.  Do NOT
build: payment features, social following, or comments.  Do not write any
code yet. First, output the proposed file structure and Supabase database
schema. Wait for my approval before continuing."
```

3.3 Advanced Prompting Techniques

The Sequential Build Prompt

Never ask the AI to build the whole app at once. The more complex your request, the more likely the AI is to hallucinate or produce inconsistent code. Build in isolated, sequential steps:

1. 'Build only the static frontend shell with dummy data first. Stop after this step.'
2. 'Now connect the frontend to the Supabase database schema we defined. Keep the same UI.'
3. 'Implement the Create and Read functionality for recipes. Leave Update and Delete for the next step.'
4. 'Add Update and Delete. Ensure the UI gives clear confirmation before deleting.'

The Context Window Prompt

When working on large codebases, be explicit about scope. AI models get confused and produce inconsistent code when you ask them to change too many files at once:

```
"We are modifying only the file components/RecipeCard.tsx. Do not touch
any other files. In this component: 1. Add a heart icon button to the
top-right of the card 2. When clicked, it should toggle a local 'saved'
state 3. Use the existing Tailwind utility classes used in this file 4.
Do not add any new dependencies"
```

The Debug Prompt

When something break and it will use a structured debug prompt. Vague complaints produce vague fixes:

```
"I am getting this error: [PASTE THE FULL ERROR MESSAGE]  Context: - The
error occurs in components/RecipeForm.tsx, around the image upload
function - It only happens when I upload a file larger than 5MB - The
```

```
Cloudinary integration was added in the last session Please: 1. Explain exactly why this error is occurring 2. Identify the specific line or function responsible 3. Provide the fixed code block 4. Do not change any other functionality in the file"
```

The Refactor Prompt

Every 10–15 prompts, your codebase accumulates technical debt. Schedule regular housekeeping:

```
"Review the file utils/recipeHelpers.ts. Refactor it to: 1. Remove any duplicate or redundant functions 2. Improve variable naming for readability 3. Ensure consistent TypeScript typing throughout 4. Add JSDoc comments to each exported function Critical: Do not change any function's behavior or output. Only improve the internal structure."
```

3.4 The Rules File - Your AI's Constitution

The single highest-leverage tool in your vibe coding arsenal is the rules file. Available in Cursor as .cursor rules and in Claude Code as CLAUDE.md, this file is read by the AI before every generation. It defines your preferences once, so you don't have to repeat them in every prompt.

```
# PROJECT RULES — Read this before every response  ## Tech Stack - Framework: Next.js 14 with App Router - Language: TypeScript (strict mode, no 'any' types) - Styling: Tailwind CSS only (no inline styles, no CSS modules) - Database: Supabase (use the existing client from lib/supabase.ts) - State: Zustand for global state, React useState for local  ## Code Style - Functional components only (no class components) - Keep components under 200 lines; extract sub-components if needed - Always handle loading and error states - Never hardcode API keys or secrets  ## What NOT To Do - Never install new dependencies without asking first - Never modify the auth logic in lib/auth.ts - Never remove existing TypeScript types to fix errors
```

> **Pro Tip: The Rules File Evolves**
>
> Start your rules file simple and add to it as you discover preferences. Every time the AI does something you didn't like, add a rule to prevent it. After a few projects, your rules file becomes a powerful context document that makes every future build faster and more consistent.

Chapter 4: The Complete Build Workflow

Building software with vibe coding is a structured, repeatable process. Here is the exact workflow that experienced vibe coders use to take a project from idea to deployed product without writing a line of traditional code.

4.1 Phase 1 — Ideation and Architecture (Before Any Code)

The biggest mistake beginners make: they jump straight to the code. Professionals spend 20% of their time here, and it saves 80% of their time later.

Step 1: Write Your Product Brief

Answer these questions in plain English before opening any tool:

- What problem does this solve, and for whom specifically?
- What are the 3–5 core features of the MVP (nothing more)?
- What does success look like after 30 days?
- What should this app never do? (Constraints are as important as features)

Step 2: Wireframe on Paper

Sketch the main screens on paper or in Figma before prompting anything. Two key screens will anchor all your UI generation: the main dashboard and the primary action form. Without this, the AI will invent an interface that may not match your mental model.

Step 3: Choose Your Stack Consciously

The choice of tech stack is a long-term commitment. Pick the simplest stack that can grow with you:

- Frontend: Next.js (React) the default choice for most web apps in 2026
- Styling: Tailwind CSS- AI generates excellent Tailwind, much better than plain CSS
- Database + Auth: Supabase instant Postgres database, built-in authentication, edge functions
- Deployment: Vercel (for Next.js) or the built-in deployment of your vibe coding tool

4.2 Phase 2 - The First Generation

Step 4: Write Your PRD Prompt

Using the framework from Chapter 3, write your full PRD prompt. Ask for the file structure and database schema first, without generating code. Review and approve before proceeding.

Step 5: Generate the UI Shell

Build the static frontend first with dummy data. This lets you see and feel the interface before you've committed to any backend logic. If it looks wrong, fix it here it's far easier to change before real data flows through it.

Step 6: Hook Up the Backend

Now connect your UI shell to the real database. This is typically: set up Supabase tables → create API routes → replace dummy data with real queries.

> ⚠ **The 70% Problem**
>
> The first generated version will capture about 70% of your vision. Expect rough edges, missing states, and UI inconsistencies. This is normal. Treat the first generation as a very functional wireframe, not a finished product. Your job from here is iterative refinement.

4.3 Phase 3 - The Iteration Loop

This is where vibe coding lives. You test, discover issues, describe fixes, test again. Keep your iterations tight: one feature at a time, one bug at a time.

The Checkpoint Ritual

After every 3–4 prompts, stop and run through this checklist:

5. Open the app in your browser
6. Test the most recent feature end-to-end
7. Try to break it: submit empty forms, use unexpected inputs, click buttons rapidly
8. Check for visual regressions: did a previous screen break?
9. If you found bugs: fix them before asking for the next feature

> 🛑 **The Cardinal Rule**
>
> NEVER ask for a new feature while a previous feature is broken. Building on broken foundations collapses the entire project. The AI will try to work around broken code, creating cascading failures that become impossible to untangle.

4.4 Phase 4 - Version Control and Safety Nets

Vibe coding makes it dangerously easy to accidentally overwrite working code. You must use version control from day one.

Git Commit Strategy for Vibe Coders

- Commit every time a feature works end-to-end, even if you plan to change it later
- Write descriptive commit messages: 'feat: add recipe creation form with image upload' not 'update'
- Create a new branch before attempting any large refactor
- If the AI makes a change that breaks the app and you can't fix it, git checkout to your last working commit

```
# Quick Git safety workflow git add -A git commit -m "feat: recipe card with save functionality working"  # Before a risky refactor: git checkout -b refactor/recipe-form  # If it goes wrong: git checkout main
```

4.5 Phase 5 - Deployment

Getting to a live URL is the goal. Modern vibe coding tools make this a single click:

- Bolt.new and Lovable: built-in one-click deployment
- Next.js projects: push to GitHub, connect to Vercel auto-deploys on every commit
- Replit: deploy button built into the UI

The rule: ship something to a real URL before it's 'finished.' Real user feedback is more valuable than another week of iteration in isolation.

PART FOUR

The Risks

What can go wrong and exactly how to prevent it

Chapter 5: Security - The Invisible Threat

This is the chapter most vibe coding guides skip. Don't skip it. The security data from 2025 - 2026 is sobering, and understanding it will save your project and potentially your users from real harm.

> ⚠ **The Sobering Statistics**
>
> • Veracode's 2025 GenAI Code Security report: 45% of AI-generated code introduces security vulnerabilities • December 2025 Code Rabbit analysis of 470 GitHub PRs: AI co-authored code had 1.7x more 'major' issues than human-written code • AI code showed 2.74x higher security vulnerabilities and 75% more misconfigurations • Roughly 24.7% of AI-generated code has a security flaw as of early 2026

5.1 The Six Major Security Risks

1. Hallucinated Security Bypasses

AI can accidentally remove a security check. It might delete an authentication keyword while refactoring. The result: pages that should be private become publicly accessible. This is not a hypothetical it has happened in production.

The Fix: Never let AI touch your authentication files without explicit review. Create a rule: 'Never modify the auth logic in lib/auth.ts.'

2. SQL Injection

AI assistants commonly generate database queries using string concatenation instead of parameterized statements. The code looks correct and passes tests but a crafted input can restructure the database query at runtime, exposing, modifying, or deleting data.

```
// DANGEROUS (AI often generates this): const query = `SELECT * FROM
users WHERE email = '${email}'`;  // SAFE (always ask for this
explicitly): const query = supabase.from('users').select('*').eq('email',
email);
```

3. Cross-Site Scripting (XSS)

AI-generated UI components frequently skip consistent output encoding. A template that looks harmless becomes a vulnerability once real user input flows through it. A malicious user can inject JavaScript that runs in other users' browsers.

The Fix: In your rules file, add: 'Always sanitize and encode user input before rendering. Never use dangerouslySetInnerHTML without explicit sanitization.'

4. Hardcoded Secrets

AI models sometimes embed API keys, database URLs, or passwords directly in code especially when you paste them in a prompt for context. Anything committed to a Git repository is potentially public forever, even if you delete it later.

The Fix: Never paste real credentials in prompts. Use environment variables for all secrets. Add .env to your .gitignore before your first commit.

5. Client-Side Security Logic

AI sometimes places security logic on the frontend, where users can inspect and manipulate it. A real-world example: a startup called Enrichlead built a lead-generation platform where the AI put all security logic on the client-side. Within 72 hours of launch, users found they could change a single browser console value to unlock all paid features. The project had to shut down.

The Fix: All authorization checks must happen on the server. The client is never trusted.

6. Prompt Injection

A real vulnerability in development tools themselves: the CurXecute vulnerability (CVE-2025-54135) allowed attackers to order the Cursor AI development tool to execute arbitrary commands on a developer's machine. A malicious comment in code being reviewed could instruct the AI to exfiltrate data.

The Fix: Be careful what external code you ask AI tools to analyze. Treat AI agent permissions like production server permissions.

5.2 The Two-Stage Security Review Process

Data from 2025 shows that a 'self-reflection' process dramatically reduces AI security vulnerabilities. For any feature involving user data, authentication, or payments:

10. Ask the AI to build the feature
11. Then give this exact prompt:

```
"You just wrote the [feature name] functionality. Now switch roles: you
are a senior security engineer. Review the code you just wrote.  Search
for: - Any SQL injection vulnerabilities - Any XSS vulnerabilities   -
Any hardcoded credentials - Any authorization logic on the client side -
Any missing input validation  Rewrite the vulnerable sections with
security best practices. Explain each change you make."
```

5.3 The Security Rules File

Add these rules to every project's .cursorrules or CLAUDE.md:

```
## Security Rules (Non-Negotiable)  - NEVER hardcode API keys, passwords, or secrets - ALWAYS use environment variables for sensitive data - ALWAYS use parameterized queries for all database operations - ALWAYS validate and sanitize user input on the server - NEVER put authorization logic in frontend components - ALWAYS use HTTPS for external API calls - NEVER use eval() or similar dynamic execution functions - ALWAYS add CORS restrictions to API routes
```

Chapter 6: Avoiding the Vibe Coding Traps

Vibe coding has well-documented failure patterns. Knowing them in advance is the difference between shipping a product and spending months in 'development hell.'

6.1 The Technical Debt Spiral

The most common way vibe coding projects fail: you build feature after feature without ever cleaning up, until the codebase is so tangled that the AI starts writing code that contradicts itself and generates circular errors.

> **The Warning Signs**
>
> • The AI starts saying 'I'm not sure how this fits with the existing code' • Adding a small feature breaks two other features • You've tried to fix the same bug three times and it keeps reappearing • The AI generates code that imports from files that don't exist

The Prevention: Schedule a refactor session every 10–15 prompts. Use the refactor prompt from Chapter 3. Think of it like oil changes skip them and the engine seizes.

6.2 Context Window Collapse

AI models have a limit to how much they can 'hold in mind' at once. When a codebase grows large or a conversation runs long, the AI starts forgetting earlier context, generating inconsistent code, or losing track of your established patterns.

Signs of context window collapse: the AI uses a different variable naming convention than before; it reimplements a function that already exists; it changes a behavior you specifically told it to keep.

The Fix: Start a new conversation. Paste your rules file at the beginning. Describe the current state of the app before asking for anything new.

6.3 Feature Creep by Prompt

It's seductively easy to add 'just one more thing' with a vibe coding tool. The result is scope creep a bloated MVP that never ships.

The Fix: Write your MVP feature list and freeze it. Create a 'Version 2' list for every feature idea that isn't on the core list. Build V1 completely before touching V2. The discipline to ship a limited product is more valuable than the ability to build a sophisticated one.

6.4 The AI Hallucination Trap

AI models sometimes confidently implement non-existent APIs, invent package names that don't exist, or describe behaviors that no library actually has. This is called hallucination, and it can waste hours of debugging before you realize the function never existed.

Watch for:

- Import statements that fail on install
- API methods that return 'undefined is not a function'
- Descriptions that sound plausible but produce no results

The Fix: When the AI mentions a specific library, function, or API, verify it exists in the official documentation before building on it.

6.5 The Productivity Paradox

Not everything is faster with AI. A 2025 Stack Overflow study found that developers who felt approximately 20% faster with AI sometimes took approximately 19% longer once debugging and cleanup were included. AI-generated code is fast to write and slow to fix when it goes wrong.

The Fix: Use AI for what it's genuinely fast at scaffolding, boilerplate, UI generation, standard CRUD operations, and documentation. Use your own judgment for novel algorithms, complex business logic, and anything involving money or user data.

PART FIVE

The Mastery

Advanced techniques, real-world applications, and the future

Chapter 7: Building Real Products - Case Studies & Templates

Abstract advice becomes real knowledge when you see it applied to specific products. This chapter walks through four complete build scenarios with the exact workflow, prompts, and decisions you'd make in practice.

7.1 Project Type 1: The Micro-SaaS

Target: A subscription tool that solves one specific problem for a niche audience. Budget: $0 in development cost. Timeline: 48 hours to MVP.

Example: Invoice Generator for Freelancers

Core features: create client profiles, generate branded PDF invoices, track payment status.

12. Write PRD: Tech stack = Next.js, Supabase, Tailwind, react-pdf for PDF generation
13. Build the data schema: clients table, invoices table, invoice_items table
14. Generate the static UI shell: client list, new invoice form, invoice preview
15. Connect to Supabase for CRUD operations
16. Add PDF export using react-pdf library
17. Add Stripe for subscriptions (use their hosted payment page - no custom payment UI)
18. Deploy to Vercel

48-Hour Reality Check

This is achievable with vibe coding in 48 hours of focused work. A traditional developer would spend 1–2 weeks on the same scope. The key: freeze the feature list at exactly these 7 items and build nothing else until V1 ships.

7.2 Project Type 2: The Internal Business Tool

Target: A tool for your own business or team that saves hours of manual work. These are the highest ROI vibe coding projects you know the problem perfectly and don't need to market to anyone.

Example: Content Calendar for a Social Media Agency

Core features: post scheduling calendar view, client accounts, post approval workflow, status tracking (Draft/Scheduled/Published).

Key advantage: you can use 'Pure Vibe' mode here since it's internal bugs are annoying but not catastrophic. Ship fast, fix as you go.

7.3 Project Type 3: The Landing Page + Waitlist

Target: Validate a business idea in 24 hours before writing a single line of product code. The fastest and most important vibe coding use case.

Workflow:

19. Prompt Bolt.new or v0 to build a landing page with hero, features, pricing, and email capture form
20. Connect to Supabase for email storage (5-minute setup)
21. Add a Stripe payment link for a 'founding member' tier
22. Deploy and run ads or post on social media
23. If 10 people pay before the product exists build it. If not kill the idea and pick a new one

> **The Lean Startup in 2026**
>
> Eric Ries' Lean Startup methodology said: build the minimum viable product and test before you invest. Vibe coding makes the 'build' part so fast that you can now run a full market validation in a day. This is the highest-leverage use of the technology.

7.4 Project Type 4: The API Wrapper Tool

Target: Build a simple, specialized interface on top of an existing API to create a product. Examples: a specialized Twitter analytics dashboard, an Airbnb pricing optimizer, a Shopify review manager.

These projects are powerful because the 'hard part' (the data) already exists. Your value is in the interface and the specific workflow you design around it.

Key consideration: read the API's terms of service before building. Some APIs prohibit reselling access.

Chapter 8: Prompt Engineering Deep Dive

This chapter contains the specific, copyable prompt templates that experienced vibe coders use. Bookmark this section.

8.1 The Complete Prompt Library

The Architecture Prompt

```
"Act as a Senior [Frontend/Backend/Full-Stack] Developer. We are building
a [describe app] for [target user].  Core features (MVP only): 1.
[Feature 1] 2. [Feature 2] 3. [Feature 3]  Tech stack: - Frontend:
[framework] - Styling: [approach] - Database: [service] - Auth: [service]
- Deployment: [platform]  Constraints: - [What NOT to build] - [Important
limitation]  Do not write code yet. Output only: 1. Proposed folder and
file structure 2. Database schema with table names and key fields 3. List
of the 5 main components you will create  Wait for my approval."
```

The Unstuck Prompt (When AI Keeps Failing)

```
"We've been trying to solve [problem] for [N] attempts. None of them have
worked. Let's reset.  Here is the current state of [filename]: [PASTE
FILE CONTENTS]  Here is the exact error we're seeing: [PASTE ERROR]
Instead of trying to fix the existing code: 1. Explain in plain English
why you think the current approach is wrong 2. Propose a completely
different approach to solving this 3. Get my approval before writing any
code"
```

The Consistency Audit Prompt

```
"Review all the components in the /components folder. Without changing
any functionality, identify: 1. Any inconsistent naming conventions 2.
Any duplicated logic that appears in multiple files 3. Any props that are
passed but not used 4. Any TypeScript types that are missing or using
'any'  Report your findings before making any changes. Wait for my
approval to proceed."
```

The Performance Prompt

```
"Review the page [page name] for performance issues. Focus on: 1. Any
unnecessary re-renders (components that update without reason) 2. Any
missing useMemo or useCallback for expensive operations 3. Any images
```

```
that are loaded but not lazy-loaded 4. Any API calls that fire on every
render instead of once on mount  Explain each issue and provide the fix.
Do not change any visual behavior."
```

The Documentation Prompt

```
"Add JSDoc comments to all exported functions in [filename]. For each
function, document: - What it does (one line) - Each parameter: name,
type, and purpose - The return value: type and meaning - One example of
calling it  Do not change any function logic. Only add comments."
```

8.2 The Meta-Prompting Strategy

One of the most powerful and under-used techniques: ask the AI to help you write better prompts.

```
"I want to build [feature]. Before you write any code, help me write a
better prompt to describe what I need. Ask me clarifying questions until
you have enough detail to write a complete, unambiguous specification.
Then show me the specification for my approval."
```

This technique forces the AI to surface assumptions and edge cases you hadn't considered, making the resulting build dramatically more accurate.

Chapter 9: From Prototype to Production

Shipping a prototype is the beginning, not the end. This chapter covers what happens when your vibe-coded app starts getting real users and real traffic.

9.1 The Production Readiness Checklist

Category	Checklist Items
Security	`[ ] All secrets in environment variables [ ] Input validation on all API routes [ ] Auth on all protected routes (server-side) [ ] Rate limiting on API endpoints [ ] SQL injection protection (parameterized queries)`
Error Handling	`[ ] 404 and 500 error pages [ ] Form validation with user-friendly messages [ ] Loading states for all async operations [ ] Empty states for lists with no data`
Performance	`[ ] Images optimized and lazy-loaded [ ] No console errors or warnings [ ] Page load under 3 seconds on mobile [ ] Core Web Vitals score above 70`
Monitoring	`[ ] Error tracking set up (Sentry or similar) [ ] Uptime monitoring configured [ ] Basic analytics in place [ ] Logging for critical operations`
Legal	`[ ] Privacy Policy page exists [ ] Terms of Service page exists [ ] Cookie consent (if serving EU users) [ ] GDPR data deletion capability`

9.2 When to Call in a Human Developer

Vibe coding gets you to approximately 70–80% of production-readiness. The remaining 20–30% often requires human expertise. Know when to make that call:

- Your app handles payments or financial data → hire a security specialist for an audit
- You're serving healthcare or legal information → regulatory compliance requires human review
- Performance has degraded and AI fixes haven't worked → a senior developer can diagnose infrastructure issues
- You're onboarding a team → codebase architecture needs a human to establish patterns
- Scaling beyond 10,000 users → database optimization and infrastructure design require expertise

9.3 Growing Beyond Vibe Coding

Many founders use vibe coding to find product-market fit, then hire developers to scale the product. This is the optimal workflow. You've validated the idea cheaply; now you invest in engineering quality.

If you're hiring developers to work on a vibe-coded codebase, be transparent about its origins. A good developer will want to refactor sections before adding major features. Budget time and budget for this it's an investment, not a waste.

Chapter 10: The Future - Agentic Engineering

We end where Karpathy pointed us in 2026: not just vibe coding, but agentic engineering the next evolution of human-AI collaboration in software development.

10.1 What Changes With Agentic Engineering

Karpathy described it precisely: 'The new default is that you are not writing the code directly 99% of the time, you are orchestrating agents who do, and acting as oversight. Engineering to emphasize that there is an art and science and expertise to it.'

The shift from vibe coding to agentic engineering is a shift from:

VIBE CODING

- One-at-a-time prompts
- Human reads and approves each change
- Focus on the feature
- Accept all, review little

AGENTIC ENGINEERING

- Multi-step autonomous agent runs
- Human sets goals and reviews outcomes
- Focus on the architecture
- Scrutinize outputs against goals

10.2 What Stays the Same

The tools change. The AI gets smarter. But the fundamental craft remains the same: you must be able to describe what you want with precision, test what you get with rigor, and make architectural decisions with judgment. The clearer your thinking, the more powerful the AI becomes as your instrument.

'The goal is to claim the leverage from the use of agents but without any compromise on the quality of the software.' Andrej Karpathy, February 2026

10.3 Skills That Will Matter Most

If AI is writing most of the code, what human skills become more valuable, not less?

- Product thinking: knowing what to build and why
- Systems thinking: understanding how components fit together architecturally
- Security judgment: recognizing when AI output is dangerous
- Communication precision: describing intent clearly enough that an AI can act on it
- Quality assessment: knowing good code from bad code, even if you didn't write it

The Irreplaceable Human

The developer who can describe, oversee, test, and make judgment calls about AI-generated code is not made obsolete by AI they are supercharged by it. The developer who can only write code and nothing else faces a genuinely uncertain future.

Appendix A: Quick Reference - The Vibe Coder's Cheat Sheet

Golden Rules

- Write a PRD prompt before touching any tool
- Build sequentially: UI shell → backend → logic → polish
- Test every feature the moment it's generated
- Fix bugs before adding features
- Commit to Git every time something works
- Refactor every 10-15 prompts
- Keep a rules file in every project
- Never put auth or security logic on the client side
- Never paste real credentials in prompts
- Ship something real before it's 'finished'

The Tool Decision Tree

- Non-technical, building a web MVP → Lovable or Bolt.new
- Developer, working on an existing codebase → Cursor or Windsurf
- Large codebase, complex refactors → Claude Code
- Learning to code → Replit
- Building only a UI component → v0 by Vercel
- Need Vercel/Next.js speed at lowest cost → Windsurf ($15/mo)

Emergency Prompts

AI is stuck in a loop:

```
"Let's start completely fresh on this feature. Ignore everything you've
built so far. Describe three different approaches we could take to solve
[problem]. Don't write code yet."
```

App is broken and you can't find why:

```
"Something in the last 3 changes broke the app. List every file that was
modified in those changes. For each file, describe what changed and
whether that change could be responsible for [the bug]."
```


Appendix B: Glossary

Vibe Coding
A software development approach where you describe what you want in natural language and let AI models generate the code. Coined by Andrej Karpathy, February 2025.

Agentic Engineering
The evolved form of vibe coding, where developers orchestrate autonomous AI agents while maintaining oversight and quality control. Karpathy's preferred term for the practice in 2026.

PRD (Product Requirements Document)
A structured description of what you're building, who it's for, and how it should behave. The most important document in any vibe coding project.

Context Window
The amount of information an AI model can 'hold in mind' at once. When a conversation or codebase gets too large, the model can lose context.

Hallucination
When an AI model confidently generates false or non-existent information — such as an API that doesn't exist or a library function that never existed.

Technical Debt
Code that works but is messy, inconsistent, or hard to maintain. Vibe coding accumulates technical debt rapidly without regular refactoring.

Rules File
A file in your project root (.cursor rules or CLAUDE.md) that defines instructions for the AI to follow in every generation. Your project's constitution.

Context Drift
When an AI starts generating code that contradicts earlier decisions because the conversation has grown too long. Fixed by starting a new session.

Parameterized Queries
A secure method of constructing database queries that prevents SQL injection by separating the query structure from user input.

Prompt Injection
An attack where malicious content in the environment (e.g., in a file being analyzed) instructs an AI agent to perform unintended actions.

Closing Note: Ship Something

The tools are the cheapest and fastest they have ever been. The barrier to entry is effectively zero. You now have the framework.

The only thing left is execution.

Don't build the perfect app. Build a real one. Get it in front of real users. Learn from real feedback. Improve. Repeat.

The difference between someone who read this book and someone who actually builds something is one PRD prompt.

Go write it.

✦ ✦ ✦

The Vibe Coding Masterclass — Updated March 2026

