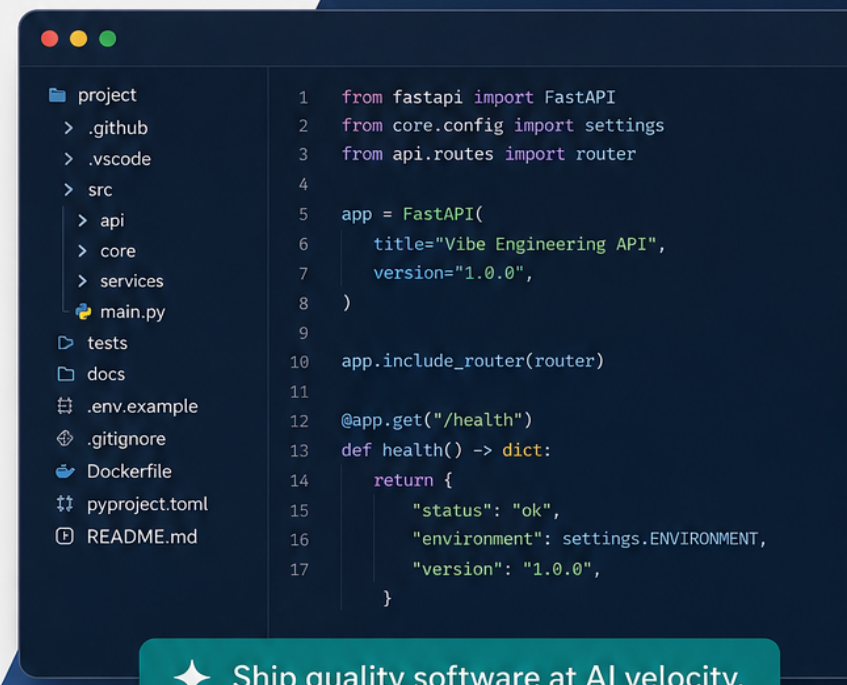
VIBE ENGINEERING

Building Production-Grade Software with AI-Assisted Development

- CONTEXT ENGINEERING
- SPEC-DRIVEN DEVELOPMENT
- AGENTIC WORKFLOWS
- SECURITY BEST PRACTICES
- TESTING STRATEGIES
- CI/CD INTEGRATION



Ship quality software at AI velocity.

GEIGER SCHMIDT

Vibe Engineering

Building Production-Grade Software with AI-Assisted Development

Steve T. Publications

This book is available at https://leanpub.com/vibeengineering

This version was published on 2026-07-13

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Contents

Building Production-Grade Software with AI-Assisted Development

This book teaches you how to build reliable, maintainable, production-grade software using AI coding assistants. You will learn context engineering, spec-driven development, agentic workflows, security practices, testing strategies, and CI/CD integration. Every chapter includes working code examples, configuration files, and step-by-step walkthroughs you can apply immediately. Whether you are a seasoned engineer looking to level up your AI workflow or a motivated beginner wanting to understand the modern development landscape, this book gives you the practical skills to ship quality software at AI velocity.

Introduction: The Vibe Engineering Revolution

What Is Vibe Engineering

In the early morning hours of February 2025, Andrej Karpathy posted a tweet that would reshape how the software industry thinks about its own craft. He described a new way of building software, one where you "fully give in to the vibes, embrace exponentials, and forget that the code even exists" [1]. He called it "vibe coding."

The term caught fire immediately. Within months, it was named Collins English Dictionary's Word of the Year for 2025 [2]. By the end of that year, Y Combinator reported that a quarter of its Winter 2025 cohort startups had codebases that were at least 95 percent AI-generated [3]. The Stack Overflow Developer Survey found that 84 percent of respondents were using or planning to use AI tools in their development process [4].

But "vibe engineering" is not the same thing as vibe coding. Vibe coding, at its most casual, means prompting an AI model and accepting whatever code comes back. It is exploratory, improvisational, and fast. It works beautifully for prototypes, scripts, and personal projects. It also produces spectacular failures when applied to production systems without discipline.

Vibe engineering is the professional practice that sits on the other end of the spectrum. It treats AI coding assistants as powerful but imperfect collaborators, requiring the same architectural rigor, testing discipline, code review standards, and security practices that any production system demands. The developer's role shifts from writing every line to orchestrating AI agents, reviewing their output, managing context, and making the judgment calls that models cannot make.

This book is about vibe engineering in that professional sense. It is about building software that works, scales, and remains maintainable over time, using AI as a force multiplier rather than a replacement for engineering fundamentals.

The Karpathy Moment and Why It Matters

Karpathy's tweet captured a cultural moment that was already underway. Large language models had reached a threshold of coding competence where they could generate working code from natural language descriptions, not just autocomplete snippets. The difference between GPT-4 in 2023 and Claude Sonnet or GPT-4.1 in 2025 is the difference between a spell-checker and a junior developer who has read the entire documentation for your framework.

This threshold crossing changed the economics of software development. If a model can write boilerplate, generate tests, scaffold APIs, and even debug errors, then the human hours required to go from idea to working prototype shrink dramatically. A solo developer who previously needed a team of three to build a full-stack application can now do it alone, faster than before.

But the threshold crossing also created new risks. When code production becomes cheap and fast, the incentives to skip testing, skip review, and skip documentation become overwhelming. The GitClear 2025 study analyzed 211 million lines of changed code from 2020 to 2024 and found that commits containing duplicated code blocks increased roughly eightfold over two years, while refactoring activity dropped from 25 percent in 2021 to under 10 percent by 2024 [5]. For the first time in the dataset's history, copy-pasted lines exceeded moved lines within a single year, a signal that developers were generating new variations of the same logic instead of extracting shared modules.

The 2025 DORA Report surveyed nearly 5,000 technology professionals and found that while over 80 percent believe AI has increased their productivity, only 24 percent trust AI-generated code heavily, and 30 percent actively distrust it [6]. This trust gap is the central tension of the AI-assisted development era. We have tools that can produce enormous quantities of code quickly, but we lack the practices to ensure that code is correct, secure, and maintainable.

Karpathy's moment matters because it forced the industry to confront this tension. You cannot "vibe" your way through building a payment processing system, a medical device controller, or any software where failure has real consequences. The skills that separate successful AI-assisted development from chaotic code generation are the same skills that have always separated good engineering from bad: clear specifications, rigorous testing, thoughtful architecture, and disciplined review. AI does not eliminate the need for these practices. It amplifies their importance.

Vibe Coding vs. Agentic Engineering

The industry has been wrestling with terminology since Karpathy's tweet. "Vibe coding" carries connotations of casualness and recklessness that many professional developers reject. Addy Osmani, an engineering leader at Google Cloud AI, wrote extensively about the distinction between vibe coding and what he calls "AI-assisted engineering" [7]. His core argument is simple: vibe coding prioritizes speed and exploration over correctness and maintainability. Professional AI-assisted development requires planning, specification, testing, and review at every stage.

Karpathy himself later suggested the term "agentic engineering" as a more accurate description of what professional developers are actually doing [8]. You are not just prompting a chatbot. You are orchestrating AI agents that can read your codebase, execute commands in a terminal, edit files across your project, run tests, and iterate on their own output. You act as the architect, reviewer, and decision-maker while the agent handles implementation details.

This book uses "vibe engineering" as an umbrella term because it captures the spirit of working with AI's generative capabilities while acknowledging the engineering discipline required to make the output production-ready. The book covers the full spectrum from casual exploration to rigorous professional practice, and you can find the level that matches your project's needs.

The key distinction to keep in mind throughout: vibe coding is a technique. Vibe engineering is a methodology. A technique can be used well or poorly. A methodology provides the structure that makes consistent, high-quality outcomes possible.

What This Book Will Teach You

This book is organized to take you from foundational concepts through advanced implementation patterns. Here is the journey:

You will start by understanding the current landscape of AI-assisted development, including the tools available, what the research says about their impact on productivity and code quality, and why there is genuine disagreement among experts about whether AI makes developers faster or slower.

Next, you will learn to build your personal tool stack. The AI coding tool market moves fast, with new products appearing every few months. You need

to understand what each major tool does well, where it falls short, and how to combine tools for maximum effectiveness.

The core of the book focuses on the practices that separate professional AI-assisted development from casual prompting. You will learn context engineering, the skill of curating the information that AI agents see so they produce better output consistently. You will learn spec-driven development, a methodology where detailed specifications serve as the bridge between your requirements and the code that AI generates. You will learn prompt strategies specifically designed for code generation, including chain-of-thought reasoning for architecture decisions and iterative refinement loops.

Quality assurance gets its own deep treatment. AI-generated code has different failure modes than human-written code, and your testing strategy needs to account for those differences. You will learn how to catch tautological tests (tests that pass because they test what the AI says the code does, not what it actually does), how to apply fuzz testing and property-based testing to AI output, and how to build integration and end-to-end coverage that catches the kinds of errors AI most commonly introduces.

Security is a major concern. AI models hallucinate package names at a rate of approximately 20 percent when generating code with dependencies, creating a new class of supply chain attacks called "slopsquatting" where malicious actors register the hallucinated package names and inject malware [9]. You will learn how to detect and prevent these attacks, along with other security risks specific to AI-generated code.

The book also covers the operational side: integrating AI assistants into CI/CD pipelines, automating code review, building self-healing pipelines, and deploying AI-assisted teams. You will learn about the Model Context Protocol (MCP), the emerging standard for connecting AI agents to external tools and data sources, and how to build custom MCP servers for your own systems.

Advanced chapters explore multi-agent orchestration, where multiple specialized AI agents work together on complex tasks, and the governance patterns needed to keep those agents coordinated and safe. The book also extends beyond web backend APIs to cover AI-assisted mobile development (React Native, Flutter), data engineering pipelines (dbt, Spark, Airflow), frontend framework integration (React, Vue, Svelte), and infrastructure-as-code workflows (Terraform, Pulumi, Kubernetes).

The book culminates in a complete project walkthrough: building

a production-grade task management API from specification through deployment, using AI-assisted development at every stage. You will see how all the practices come together in a real project, with working code examples, configuration files, and repository structures you can adapt for your own work.

Throughout, the emphasis is on practical, actionable guidance. Every chapter includes concrete examples you can follow along with. The goal is not to give you a theoretical understanding of AI-assisted development but to equip you with the skills to use it effectively in your daily work.

The rest of this book assumes one thing: that AI-assisted development is here to stay, and the developers who learn to use it well will have a significant advantage. The question is not whether to adopt these tools but how to do it right. That is what vibe engineering is all about.

Chapter 1: The AI-Assisted Development Landscape

From Autocomplete to Agentic Coding

The story of AI in software development begins with autocomplete. Tools like Tabnine, launched in 2017, and GitHub Copilot, released in June 2021, offered line-by-line code suggestions based on patterns learned from billions of lines of open-source code. They were helpful but limited: the model could suggest the next few lines if you had already written enough context for it to understand what you were doing. You still needed to write the first function, the first class, the first file.

The breakthrough came with large language models that could generate entire files, entire modules, and eventually entire projects from natural language descriptions. GPT-4, released in March 2023, demonstrated coding capabilities that surprised even its creators. It could read a bug report, understand the relevant code, and propose a fix. It could translate requirements into working implementations. It could explain complex code to beginners and refactor legacy systems to modern patterns.

But GPT-4 was still a chatbot. You had to copy and paste code back and forth between the model and your editor. The real productivity unlock came when AI became embedded in the development environment itself. Cursor, launched in March 2023 as a VS Code fork with AI baked into every layer, showed what was possible when the model could see your entire project, edit any file, run terminal commands, and iterate on its own output [10]. Claude Code, released by Anthropic in April 2025 as a terminal-native agent, took the concept further by operating autonomously: you could ask it to implement a feature, and it would read files, plan the changes, edit multiple files, run tests, and report back [11].

This evolution from autocomplete to agentic coding represents a fundamental shift in the developer's relationship with their tools. Autocomplete is a

suggestion engine. An agent is a collaborator. The difference matters because it changes what you ask the tool to do and how you verify its output.

The timeline of this evolution is remarkably compressed:

Year	Milestone
2017	Tabnine launches, offering AI-powered autocomplete
2021	GitHub Copilot released, bringing GPT-3 to code completion
2023	Cursor launches as AI-native IDE; GPT-4 demonstrates file-level code generation
2024	Devin, the first "AI software engineer," demos end-to-end task completion
2025	Claude Code releases as terminal agent; Karpathy coins "vibe coding"; AI reaches 90% developer adoption
2026	Multi-agent orchestration becomes mainstream; MCP ecosystem matures; spec-driven development gains industry adoption

We are still early in this evolution. The tools of 2026 would seem magical to the developers of 2023, and the tools of 2028 will likely seem equally advanced to us. The key insight is that the capabilities are improving faster than the practices. This book exists to help close that gap.

The Tool Landscape: IDEs, CLIs, and Platforms

The AI coding tool ecosystem has fragmented into several categories, each optimized for different workflows. Understanding this landscape is essential for building your own tool stack.

Standalone AI IDEs. These are complete development environments built from the ground up with AI as a first-class citizen. Cursor is the market leader, having reached $1 billion in annualized revenue in under two years [12]. It is a fork of VS Code, so all existing extensions work, but every interaction is augmented by AI. You can chat with your codebase, ask for multi-file edits, run commands through an AI-controlled terminal, and use "Composer" mode for complex, cross-file changes. Windsurf (formerly Codeium) offers

a similar experience with its "Cascade" agentic engine and a generous free tier. Google's Antigravity IDE, launched in November 2025, brings Gemini models into a dedicated editor [13]. Amazon's Kiro, released in July 2025, emphasizes spec-driven development with built-in support for writing and validating specifications before generating code [14].

IDE Extensions. These are AI assistants that plug into existing editors like VS Code, JetBrains IDEs, and Vim. GitHub Copilot remains the most widely distributed option, reaching 15 million developers by early 2025 [15]. It excels at inline autocomplete and chat-based assistance. Continue is a popular open-source extension that supports multiple model providers. Cline offers a free, open-source agentic coding experience where you bring your own API key. Augment Code provides deep codebase understanding through its "Junie" agent, launched in April 2025, which can generate tests matching existing patterns and conventions [16].

Terminal Agents. These are command-line tools that operate autonomously on your codebase. Claude Code is the most prominent example, released by Anthropic in April 2025 and surpassing $2.5 billion in annualized revenue within a year, the fastest product ramp in enterprise software history [17]. It reads your project files through a CLAUDE.md configuration, plans changes, edits files, runs commands, and iterates until the task is complete. OpenAI's Codex CLI, released in April 2025, offers similar capabilities with GPT-4.1. Aider is a popular open-source terminal agent that supports multiple model providers and has been around since June 2023 [18].

Browser-Based Platforms. These are fully cloud-hosted development environments where AI does most of the heavy lifting. Replit Agent can scaffold an entire application from a description, handling everything from code generation to deployment. Bolt (by StackBlitz) and Lovable offer similar browser-first experiences optimized for rapid prototyping. These platforms excel at getting from idea to working prototype in minutes but lack the version control, refactoring capabilities, and production hardening features that IDE-native tools provide.

Cloud Agents. Devin, launched by Cognition Labs in March 2024, was marketed as the world's first "AI software engineer." It operates in its own cloud environment with a browser, terminal, and editor. OpenHands (formerly OpenDevin) offers an open-source alternative. These platforms are still experimental and best suited for well-defined tasks rather than complex, multi-file development work.

No single tool is universally best. The most effective developers combine tools based on the task at hand: Cursor or Windsurf for daily editing, Claude Code for complex refactoring and multi-file changes, Copilot for inline autocomplete in JetBrains IDEs, and browser platforms for quick prototypes. Chapter 2 dives deeper into each category.

What the Data Says About Productivity

The question of whether AI coding assistants actually make developers more productive has generated enormous amounts of research, opinion, and conflicting data. The reality is nuanced.

Self-reported productivity gains are overwhelming. The 2025 DORA Report found that over 80 percent of respondents believe AI has increased their productivity [19]. A separate survey by the Pragmatic Engineer found approximately 85 percent of respondents using at least one AI tool in their workflow [20]. The Stack Overflow 2025 Developer Survey reported that 74 percent of developers experience increased productivity when using AI-assisted approaches [21].

Controlled studies tell a more complicated story. In July 2025, the nonprofit Model Evaluation and Threat Research (METR) published a randomized controlled trial that found experienced open-source developers were 19 percent slower when using AI coding assistants compared to working without them [22]. The study tracked 16 developers with moderate AI experience as they completed 246 real-world tasks on mature repositories averaging over one million lines of code. The developers initially expected a 24 percent speedup and maintained a perceived 20 percent gain even after the study revealed the slowdown.

This finding was not an outlier. A separate controlled trial by Becker et al. found that experienced developers were 19 percent slower when using AI coding assistants compared to working without them, specifically on complex tasks in unfamiliar codebases [23].

The resolution lies in task-model fit. The METR study's authors and subsequent analysts noted that the slowdown was most pronounced for tasks requiring deep understanding of existing codebases, complex debugging, and architectural decisions. For boilerplate generation, simple CRUD operations, test writing, and documentation, AI consistently sped things up. The issue

is that experienced developers tend to work on harder problems, where the model's limitations are more apparent.

The Google Cloud 2025 DORA Report offers a useful framing: AI acts as a "mirror and a multiplier." It amplifies what is already there. Strong teams with good processes get stronger. Teams with weak processes get exposed. The report found that organizations with mature engineering practices saw the greatest AI benefits, while organizations that used AI to compensate for poor fundamentals often made things worse [24].

Productivity metrics are being redefined. Traditional metrics like lines of code become meaningless when estimates suggest roughly 41 percent of all global code is now AI-generated or AI-assisted, representing approximately 256 billion lines written in 2024 alone [25]. (This figure comes from GitClear's analysis of repository patterns and should be treated as an estimate: more conservative studies of merged production code place AI-authored content closer to 27 percent of commits.) The DORA community has expanded its framework from four core metrics to include AI-specific KPIs like "AI Code Share" (the percentage of code attributed to AI generation), "Code Turnover Rate" (how quickly AI-generated code is modified or reverted), and "Complexity-Adjusted Throughput" [26].

The practical takeaway: AI does make developers faster at many tasks, but the gains are not uniform. The net effect on a team's productivity depends on how well the tools match the work, how disciplined the development process is, and whether the organization invests in testing and review practices that catch AI errors before they reach production.

The Code Quality Debate

If productivity is contested, code quality is where the debate gets heated. The evidence suggests that AI-assisted development, when practiced without discipline, degrades code quality. When practiced with appropriate safeguards, it can maintain or even improve it.

The GitClear findings are alarming. Their 2025 study analyzed 211 million changed lines of code from 2020 to 2024 across private repositories and 25 of the largest open-source projects [27]. Key findings include:

- Refactoring activity dropped from 25 percent in 2021 to under 10 percent by 2024

- Commits containing duplicated code blocks increased roughly eightfold over two years
- For the first time in the dataset's history, "copy/paste" operations exceeded "moved" operations within a single year (2024)
- Code churn (lines reverted or updated within two weeks of being written) rose from 3.1 percent to 5.7 percent, a 39 percent increase in AI-heavy projects

These metrics suggest that AI is encouraging developers to generate new variations of existing logic rather than extracting shared, reusable modules. The result is codebases that grow larger and more duplicated over time.

Security vulnerabilities are elevated. Veracode found security flaws in 45 percent of AI-generated code samples [28]. Endor Labs reported that 80 percent of AI-suggested dependencies carry known risks, and nearly half of those that exist contain known CVEs [29]. A study analyzing 576,000 code samples from 16 different LLMs found that approximately 20 percent of package dependencies referenced by AI models do not actually exist in package registries [30]. This hallucination rate is not random noise: 58 percent of the time, a hallucinated package name is repeated across multiple generations, making it a predictable target for malicious actors.

But the picture is not uniformly negative. The 2025 DORA Report found that 59 percent of respondents observed positive impacts on code quality when using AI [31]. The difference between these findings comes down to process. Teams that treat AI output as junior developer code, subject to the same testing, review, and security scanning standards, see quality outcomes comparable to or better than fully human-written code. Teams that accept AI output without scrutiny accumulate technical debt rapidly.

The open-source ecosystem is under pressure. A January 2026 paper argued that AI-assisted development harms open-source ecosystems by reducing maintainer engagement and homogenizing library selection [32]. GitHub reported in early 2026 that AI contributions are overwhelming maintainers, prompting the introduction of new repository controls to manage the volume of AI-generated pull requests [33].

The code quality debate is not resolved, and it may never be fully resolved because the answer depends on how you use the tools. This book takes the position that quality is a choice, not an inevitability. The practices described in the chapters ahead are designed to help you make the right choice.

Chapter 2: Your AI Development Toolkit

IDE-Native Assistants: Cursor and Windsurf

Cursor and Windsurf are the two leading AI-native IDEs. Both are built on the VS Code codebase (Code OSS), which means they support the entire VS Code extension ecosystem. The difference is in how deeply AI is integrated into the editing experience.

Cursor is the market leader, having reached $1 billion in annualized revenue by November 2025 and $2 billion by March 2026 [34]. It is not an extension on top of VS Code; it is a complete rebuild where AI is woven into every interaction. When you open a file, Cursor has already indexed your entire project. When you type, it offers completions that understand cross-file context. When you open the chat panel, the model can see your project structure, read any file, and propose edits across multiple files simultaneously.

Cursor's key features include:

- **Composer mode**: A multi-file editing interface where you describe a change in natural language and Cursor plans the modification, edits files, and shows you a unified diff before you apply it. This is Cursor's standout feature for complex refactoring.
- **Codebase indexing**: Cursor builds a vector index of your project, enabling semantic search across all files. When you ask "how does authentication work?", it finds the relevant files and summarizes the flow.
- **Terminal integration**: The built-in terminal can be controlled by AI. You can ask Cursor to run a command, interpret the output, and fix errors.
- **Tab autocompletion**: Inline suggestions that span multiple lines and understand project-wide context, powered by Claude 3.5 Sonnet or your chosen model provider.
- **Bring Your Own Model (BYOM)**: Cursor supports multiple model providers including Anthropic Claude, OpenAI GPT, Google Gemini, and

local models through Ollama. You can choose different models for chat versus autocomplete.

Cursor pricing starts at $20/month for the Pro plan with unlimited AI usage. The Business plan is $40/month per user with team management features. The Enterprise plan is custom-priced with SSO and advanced security controls.

Windsurf, formerly Codeium, went through a complex acquisition saga in 2025 [35]. After an attempted $3 billion acquisition by OpenAI fell through in April 2025, Google DeepMind executed a $2.4 billion licensing deal that brought Windsurf's CEO and key engineers to Google. Cognition AI (the Devin team) then acquired the remaining Windsurf IP and product for approximately $250 million in December 2025. It offers a similar AI-native editing experience with its "Cascade" agentic engine. Windsurf differentiates itself with:

- **Deep Research**: A feature that lets the AI investigate your codebase thoroughly before proposing changes, reading through related files and understanding dependencies.
- **Flow mode**: An immersive editing experience where AI suggestions appear inline as you type, minimizing context switching between chat and editor.
- **Generous free tier**: Windsurf offers a free plan with substantial AI usage limits, making it accessible for individual developers and small teams.
- **Cascade agent**: A built-in coding agent that can execute multi-step tasks, similar to Claude Code but integrated into the IDE.

Windsurf pricing starts free for individual developers, with the Pro plan at $15/month and the Business plan at $20/month per user.

When to use Cursor versus Windsurf: Choose Cursor if you want the most mature AI-native IDE experience, the best multi-file editing capabilities, and are willing to pay a premium. Choose Windsurf if you want strong AI features at a lower cost, particularly the free tier for individual work. Many professional developers use both: Cursor as their primary editor and Windsurf for quick tasks on personal projects.

Terminal Agents: Claude Code and OpenAI Codex CLI

Terminal agents represent a fundamentally different interaction model from IDE assistants. Instead of working within the editor, they operate from the

command line, reading files, editing code, running commands, and iterating autonomously. You give them a task description, and they execute it.

Claude Code is Anthropic's terminal-native coding agent, made generally available in May 2025 [36]. It has grown faster than any product in enterprise software history, reaching $1 billion in annualized revenue by November 2025 and surpassing $2.5 billion by February 2026 [37]. Claude Code is built on the Claude Sonnet model and supports the full range of Claude's coding capabilities.

Key features:

- **Autonomous multi-file editing**: Claude Code can read your entire project, plan changes across multiple files, implement them, run tests, and iterate on failures. You describe what you want in natural language, and it handles the implementation.
- **CLAUDE.md configuration**: A project-level file that tells Claude Code about your project's conventions, architecture, testing approach, and any specific rules it should follow. This is loaded automatically at the start of every session.
- **Skills**: Reusable prompt templates that encode common workflows. Anthropic provides built-in skills for tasks like writing tests, debugging, and documentation, and you can create custom skills for your team's patterns.
- **Subagents**: Claude Code can spawn specialized sub-agents for deep tasks (like analyzing a large database schema) while maintaining the main conversation. Sub-agents return condensed summaries to preserve context.
- **MCP server integration**: Claude Code connects to Model Context Protocol servers, giving it access to databases, APIs, documentation systems, and other external tools.
- **Hooks**: Deterministic automation that runs at specific points in the workflow (before editing, after generating tests) without consuming model tokens.

Claude Code pricing is $20/month for the Pro plan and $200/month for the Max plan. Usage is measured in input/output tokens, and heavy users can consume credits quickly.

OpenAI Codex CLI was released in April 2025 as OpenAI's terminal-native coding agent [38]. It is built on GPT-4.1 and offers similar capabilities to Claude Code:

- **Autonomous task execution**: Describe a coding task and Codex reads files, edits code, runs commands, and iterates.
- **Multi-model support**: Codex can use different GPT models for different phases of a task (reasoning for planning, faster models for implementation).
- **MCP integration**: Like Claude Code, Codex connects to MCP servers for external tool access.
- **Structured outputs**: Built-in support for OpenAI's structured outputs API, ensuring generated code conforms to specified JSON schemas.

Codex CLI pricing starts free with usage limits, with paid plans scaling up to $200/month for heavy usage.

Aider is a popular open-source terminal agent that has been around since June 2023 [39]. It supports multiple model providers through its own API abstraction layer. Key strengths include:

- **Open-source and self-hostable**: You can run Aider locally with any model provider, including local models through Ollama.
- **Git integration**: Aider creates commits for each change it makes, making it easy to review and revert.
- **Flexible model support**: Works with Claude, GPT-4, Gemini, and open-source models.

Aider is free to use; you only pay for your API calls to the model provider.

When to use terminal agents: Terminal agents excel at tasks that involve reading many files, making coordinated changes across a codebase, and iterating through test failures. They are particularly effective for refactoring, implementing features from specifications, writing tests, and debugging complex issues. Many developers pair a terminal agent with an IDE assistant: Claude Code or Codex CLI for the heavy lifting, Cursor or Copilot for day-to-day editing.

Inline Autocomplete: GitHub Copilot and Alternatives

Inline autocomplete tools sit between your keyboard and the editor, suggesting code as you type. They are the least intrusive form of AI assistance and the

easiest to adopt because they require no change to your workflow. You either accept the suggestion (usually with Tab) or keep typing.

GitHub Copilot is the most widely distributed autocomplete tool, reaching 15 million developers by early 2025 [40]. It is available as an extension for VS Code, JetBrains IDEs, Visual Studio, Neovim, and other editors. Key features:

- **Inline completions**: Multi-line suggestions that appear as ghost text in your editor. Copilot uses a combination of fast local models for immediate suggestions and cloud-based models for higher-quality completions.
- **Chat panel**: A side-panel chat where you can ask questions about your code, request refactoring, or generate new code.
- **Copilot Agent**: An agentic mode (released May 2025) that can execute multi-file tasks, similar to Cursor's Composer but within the Copilot interface.
- **GitHub integration**: Deep integration with GitHub repositories, issues, and pull requests. Copilot can reference your project's issues and PR descriptions to provide more relevant suggestions.
- **Multi-model support**: Copilot uses a mix of models including GPT-4, Claude, and custom OpenAI models, automatically selecting the best model for each task.

Copilot pricing is $19/month for individuals, $19/month per user for Business, and $39/month per user for Enterprise. Students and maintainers of popular open-source projects get it free.

Continue is an open-source VS Code extension that supports multiple model providers [41]. It offers inline completions, chat, and a customizable configuration. Key advantages:

- **Provider flexibility**: Works with any model provider, including local models. You can configure different models for completions versus chat.
- **Custom rules**: Define project-specific rules that guide the AI's suggestions.
- **Open-source**: Full transparency about how your code is processed and sent to models.

Continue is free to use; you pay only for API calls to your chosen model provider.

Tabnine was one of the first AI autocomplete tools, launched in 2017 [42]. It offers both cloud-based and fully local (on-device) code generation, making it attractive for organizations with strict data privacy requirements. Tabnine's Enterprise plan supports fully on-premises deployment.

When to use inline autocomplete: Autocomplete is best for routine coding tasks where you already know what you want to write and the model can save you keystrokes. It is less effective for complex architectural decisions, cross-file refactoring, or understanding unfamiliar codebases. Most professional developers use autocomplete as a baseline tool and switch to IDE assistants or terminal agents when the task requires deeper reasoning.

Browser-Based Platforms: Replit, Bolt, and Lovable

Browser-based platforms take a radically different approach. Instead of installing software on your machine, you work entirely in the browser. The AI scaffolds an entire application from a natural language description, handles the build process, and deploys the result to the cloud. These platforms are optimized for speed: going from idea to working prototype in minutes rather than hours.

Replit Agent is Replit's AI-powered development assistant [43]. You describe what you want to build, and the Agent creates the project structure, writes the code, installs dependencies, and deploys the application. Key features:

- **Full-stack scaffolding**: Can generate frontend, backend, database schema, and deployment configuration from a single description.
- **Integrated hosting**: Applications are deployed directly to Replit's cloud infrastructure.
- **Collaborative editing**: Multiple users can work on the same project simultaneously with AI assistance.
- **Iterative refinement**: You can describe changes in natural language, and the Agent modifies the code accordingly.

Replit pricing starts free with limited resources, with paid plans from $7/month (Core) to $24/month (Teams).

Bolt by StackBlitz is a browser-based development environment optimized for web applications [44]. It runs Node.js entirely in the browser using WebContainers technology, meaning the full development stack executes client-side without a server. Key features:

- **Instant project creation**: Describe a React, Next.js, or Vue application and Bolt generates the code and runs it immediately in the browser.
- **No setup required**: No need to install Node.js, configure build tools, or set up local environments.
- **One-click deployment**: Deploy to Netlify, Vercel, or Cloudflare Pages directly from the interface.

Bolt offers a free tier with usage limits and paid plans starting at $15/month.

Lovable is a browser-based platform that focuses on full-stack web applications [45]. It generates both frontend and backend code, handles database schema creation, and deploys to the cloud. Lovable has been in the news for security issues in generated code, including publicly accessible databases and hardcoded credentials [46].

When to use browser platforms: Browser platforms excel at rapid prototyping, proof-of-concept development, and learning new frameworks. They are not suitable for production applications that require careful architecture, thorough testing, and integration with existing systems. Use them to explore ideas quickly, then move to IDE-native tools for serious development work.

Building Your Tool Stack

No single tool covers every use case. The most effective developers build a tool stack that combines complementary tools for different phases of the development process. Here are three common configurations:

The Solo Developer Stack:

- Primary editor: Cursor ($20/month)
- Terminal agent: Claude Code Pro ($20/month)
- Total: $40/month

This combination gives you the best IDE experience for daily editing and a powerful terminal agent for complex tasks. It is the most common configuration among professional indie developers.

The Budget-Conscious Stack:

- Primary editor: Windsurf (free tier or $15/month Pro)
- Terminal agent: Aider (free, pay per API call)
- Inline autocomplete: Continue (free, pay per API call)
- Total: $0 to $15/month plus API costs

This stack is fully functional for individual developers who are comfortable managing their own API keys and choosing models based on cost-performance tradeoffs.

The Enterprise Stack:

- Primary editor: Cursor Business ($40/month per user)
- Terminal agent: Claude Code Max ($200/month per user) or self-hosted Aider
- Inline autocomplete: GitHub Copilot Enterprise ($39/month per user)
- CI/CD integration: AI code review tools (CodeRabbit, Qodo, or Ellipsis)
- Total: $279+/month per user plus infrastructure costs

Enterprise teams add overhead for security, compliance, and team management but get the most comprehensive tooling.

Open-source alternatives: If you prefer to avoid vendor lock-in, you can build a fully open-source stack using Continue or Cline as your IDE assistant, Aider as your terminal agent, and local models through Ollama or LM Studio. The tradeoff is that local models are generally less capable than cloud-based frontier models, particularly for complex reasoning tasks.

The key principle is to choose tools that match your workflow, not the other way around. If you already use JetBrains IDEs and love them, there is no reason to switch to Cursor just because it has better AI features. GitHub Copilot works well in IntelliJ and PyCharm. If you are a terminal person who rarely opens a GUI editor, Claude Code or Aider may be all you need. The best tool stack is the one you actually use consistently.

Chapter 3: Context Engineering

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Why Context Is Your Most Valuable Resource

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

The Anatomy of a Great CLAUDE.md

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Token Budgets and Attention Decay

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Just-in-Time Retrieval and Progressive Disclosure

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Structured Note-Taking and External Memory

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Chapter 4: Prompt Strategies for Code Generation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

The Spec-First Prompt Pattern

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Chain-of-Thought for Architectural Decisions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Few-Shot Examples and Negative Constraints

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Iterative Refinement Loops

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Model Selection by Task Type

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Chapter 5: Spec-Driven Development

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

From Vibe Coding to Specs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Writing Effective Specifications

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

API Endpoints

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Non-Functional Requirements

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Out of Scope (for this iteration)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

The Plan-Build-Verify Cycle

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Tools That Support Spec-Driven Workflows

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Chapter 6: Architecture and Design for AI-Assisted Development

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Designing for AI Generation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Modular Monoliths and Bounded Contexts

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

API-First Architecture

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Event-Driven Patterns and Async Workflows

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Anti-Patterns to Avoid

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Chapter 7: Testing AI-Generated Code

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Why AI-Generated Code Needs Different Testing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Test Generation with AI Assistants

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Catching Tautological Tests

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Fuzz Testing and Property-Based Testing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Integration and End-to-End Coverage

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Troubleshooting: Common AI Testing Pitfalls

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

The Testing Checklist for AI Code

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Chapter 8: Debugging Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

The Five-Step Debugging Methodology

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Common Failure Patterns in AI-Generated Code

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Production Debugging with AI Agents

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Observability and Distributed Tracing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

When to Fall Back to Classical Debugging

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Chapter 9: Security and Supply Chain

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

The Hallucination Attack Surface

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Slopsquatting and Supply Chain Risks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Case Study: Slopsquatting Incident Post-Mortem

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Injection Vulnerabilities in AI Code

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Secrets Management and Credential Safety

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Security Scanning in the AI Pipeline

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Chapter 10: CI/CD Integration

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

The AI-Native CI/CD Pipeline

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Automated Code Review with AI

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

AI-Augmented Test Generation in CI

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Self-Healing Pipelines

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Deployment Strategies for AI-Assisted Teams

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Knowledge Sharing and Context Propagation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Onboarding and Team Standards

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Measuring What Matters: DORA Metrics and Beyond

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Chapter 12: The Model Context Protocol (MCP)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

What Is MCP and Why It Matters

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

The Client-Server Architecture

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Building Custom MCP Servers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Security and Authorization in MCP

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

The Growing Ecosystem

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Chapter 13: Multi-Agent Orchestration

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

From Single Agent to Multi-Agent Systems

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Orchestration Patterns: Planner, Supervisor, Hive Mind

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Frameworks: LangGraph, CrewAI, Microsoft Agent Framework

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Parallel Execution and Worktree Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Governance and Guardrails

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Chapter 14: Real-World Project Walkthrough

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Project Brief: A Task Management API

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Phase 1: Specification and Planning

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Phase 2: Scaffold and Architecture

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Phase 3: Implementation with Agents

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Phase 4: Testing and Validation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Phase 5: Deployment and Monitoring

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Chapter 15: AI-Assisted Development Across Domains

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Mobile Development with AI Assistants

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Data Engineering Pipelines with AI

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Frontend Framework Integration

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Infrastructure as Code and DevOps with AI

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Cross-Domain AI Workflow Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Chapter 16: Maintenance, Migration, and Evolution

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Refactoring AI-Generated Code

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Legacy System Migration

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Performance Optimization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Managing Technical Debt at Scale

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Keeping Your Codebase AI-Friendly

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Conclusion: The Future of Vibe Engineering

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

The Skills That Will Endure

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

What Comes Next

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

Your Role in the AI-Native Future

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.

References

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/vibeengineering.