

In a Cave, With a Box of Scraps: A Thesis on Solo Building with Vibe Coding in the Pre-Jarvis Era

Will Glynn February 2025

Abstract

This paper examines the practice of "vibe coding"—the emerging methodology of building complex software systems through natural language collaboration with large language models—and argues that despite its current limitations, early adopters are laying the groundwork for a fundamental transformation in software development. Drawing parallels to Tony Stark's construction of the first Iron Man suit under impossible constraints, we explore why building with imperfect AI tools today is not merely an exercise in frustration, but a necessary phase of technological evolution that will ultimately yield superhuman development capabilities.

I. Introduction: The Cave

"Tony Stark was able to build this in a cave! With a box of scraps!" — Obadiah Stane, *Iron Man* (2008)

The line is meant as an insult, a dismissal of lesser engineers who cannot replicate Stark's genius. But embedded within it is a profound truth about innovation: sometimes the most transformative technologies are born not in pristine laboratories with unlimited resources, but in caves, with scraps, under pressure, by those stubborn enough to believe that constraints are merely suggestions.

In 2025, those of us building with AI-assisted development tools—what has come to be known as "vibe coding"—find ourselves in our own cave. The scraps we work with are large language models that hallucinate, lose context, suggest deprecated APIs, and occasionally produce code that would make a junior developer wince. Our cave is the gap between what AI *could* be and what it *currently is*: a brilliant but unreliable partner prone to confident mistakes.

And yet, we build anyway.

This thesis argues that the current era of vibe coding, despite its substantial drawbacks, represents a critical formative period in the history of software development. Those who persist through the debugging sessions, the context window limitations, and the maddening loops of AI confusion are not merely tolerating suboptimal tools—they are training themselves for a future where AI capabilities will be orders of magnitude more powerful, and where the ability to collaborate with machine intelligence will be the primary differentiator between developers.

II. Defining Vibe Coding

Vibe coding is the practice of describing desired software behavior in natural language and iteratively refining the output through conversation with an AI system. Unlike traditional programming, where the developer must translate their intent into precise syntactic instructions, vibe coding allows for high-level specification: "make the modal flow from welcome to wallet creation without race conditions" rather than manually refactoring state management.

The methodology emerges from a recognition that programming has always been about *intent translation*. The evolution from machine code to assembly to high-level languages to frameworks has consistently moved toward allowing developers to express what they want at higher levels of abstraction. Vibe coding is the logical continuation: expressing intent in the highest-level language available—human natural language—and delegating the translation to an AI system.

Characteristics of Vibe Coding

1. **Conversational Development:** Code emerges through dialogue rather than solitary typing
 2. **Intent-First Specification:** Focus on what the software should do, not how it should be implemented
 3. **Rapid Prototyping:** Ideas can be tested in minutes rather than hours
 4. **Context Dependency:** The AI's understanding of the codebase shapes its suggestions
 5. **Collaborative Debugging:** Errors are diagnosed through discussion, not just stack traces
-

III. The Current Limitations: Why We're Still in the Cave

A. The Context Window Problem

Current LLMs operate within fixed context windows—a limited amount of text they can "remember" at once. For complex projects like VibeSwap, with its interconnected smart contracts, React frontend, Python oracle, and cross-chain messaging layer, no single conversation can hold the entire system in mind.

The result is an AI that forgets. It forgets the wallet security axioms you established. It forgets the pattern you've been using for dual wallet detection. It forgets that you just fixed this exact bug three messages ago. The developer becomes a context manager, constantly reminding the AI of information it should already know.

B. Hallucination and Confident Incorrectness

LLMs do not know what they do not know. When asked about an unfamiliar API, they will not say "I'm uncertain." They will confidently generate plausible-looking code that uses functions that don't exist, parameters that were deprecated three versions ago, or patterns that are subtly wrong in ways that only manifest at runtime.

The debugging overhead this creates is substantial. A simple feature that should take ten minutes can balloon into an hour of tracing why the AI's suggested solution doesn't work, discovering the hallucinated method, finding the correct approach, and then guiding the AI back to a working implementation.

C. The Loop of Confusion

Perhaps the most frustrating pattern in vibe coding is the loop: the AI makes a mistake, you correct it, it overcorrects in a different direction, you correct again, it reverts to the original mistake, and suddenly you're four messages deep in a circular argument with a machine that has lost the thread of what you're even trying to accomplish.

The user's words echo in memory: *"not to be too clever."* Sometimes the AI's attempt to be sophisticated—to anticipate edge cases, to add defensive programming, to refactor while fixing—creates more problems than it solves. Simplicity becomes a discipline that must be constantly enforced.

D. The Solo Amplification Effect

For solo developers, these limitations are amplified. There is no teammate to review the AI's suggestions, no second opinion when the model insists its solution is correct, no human backstop when exhaustion clouds judgment. The solo vibe coder must simultaneously be:

- The product manager defining requirements
- The architect ensuring systemic coherence
- The developer implementing features
- The QA engineer catching AI mistakes
- The DevOps engineer deploying and monitoring
- The AI handler managing context and correcting course

This multidimensional cognitive load is the true cost of building in the cave.

IV. Why We Build Anyway: The Jarvis Thesis

A. The Trajectory is Clear

The capabilities of large language models have improved at an exponential rate. GPT-3 to GPT-4 represented a qualitative leap in reasoning. Claude's evolution has shown similar trajectories in code understanding and generation. The pattern is unmistakable: what is frustratingly limited today will be remarkably capable tomorrow.

The Jarvis Thesis states: *Within the foreseeable future, AI development assistants will achieve a level of capability where they can autonomously handle complex software engineering tasks with minimal human oversight—understanding entire codebases, maintaining perfect context, making zero hallucination errors, and anticipating developer needs before they are expressed.*

This is not science fiction. It is the obvious extrapolation of current trends. The question is not *if* but *when*.

B. Early Adoption as Competitive Advantage

Those who learn to work with AI tools in their primitive state develop skills that will be invaluable when the tools mature:

1. **Prompt Engineering Intuition:** Understanding how to communicate effectively with AI systems
2. **AI Output Evaluation:** Quickly assessing the quality and correctness of generated code
3. **Human-AI Workflow Design:** Knowing when to delegate to AI and when to intervene
4. **Context Management:** Strategies for maintaining coherent project state across conversations
5. **Debugging AI Mistakes:** Pattern recognition for common AI failure modes

These skills are being forged in the cave, under pressure, with imperfect tools. When Jarvis arrives, those who have been building Iron Man suits with scraps will be ready to build the Mark L.

C. The Existence Proof

Every successful project built with vibe coding is an existence proof—evidence that the methodology works despite its limitations. VibeSwap, with its:

- Commit-reveal batch auction smart contracts
- LayerZero V2 cross-chain messaging
- Kalman filter price oracle
- WebAuthn device wallet integration
- React frontend with complex modal state management
- Shapley value reward distribution

...all built substantially through natural language collaboration with AI, demonstrates that vibe coding can produce production-grade software. The debugging was painful. The context management was tedious. The hallucinations were maddening. But the system *works*.

Each such proof normalizes the methodology and paves the way for broader adoption. Each struggle documents patterns for future developers. Each success moves us closer to the day when the cave becomes a penthouse laboratory.

V. The Philosophy of Constraint

There is a deeper argument for building with suboptimal tools: constraints breed creativity.

Tony Stark didn't build the Mark I because a cave was the ideal workshop. He built it because he had no choice, and the pressure of mortality focused his genius. The resulting design—crude, improvised, barely functional—contained the conceptual seeds of every Iron Man suit that followed. The arc reactor, the flight stabilizers, the weaponized repulsors: all born in a cave, with a box of scraps.

Similarly, the patterns we develop for managing AI limitations today may become foundational for AI-augmented development tomorrow:

- **Explicit Context Documentation:** CLAUDE.md files that maintain project state across sessions may evolve into sophisticated AI memory systems
- **Incremental Verification:** The habit of testing AI output immediately may become automated validation pipelines
- **Intent Specification Languages:** Our natural language prompts may formalize into structured intent schemas
- **Human-AI Pair Programming Protocols:** Our ad-hoc collaboration patterns may become standardized methodologies

We are not just building software. We are building the practices, patterns, and mental models that will define the future of development.

VI. The Struggle as Selection

Not everyone can build in a cave. The frustration, the setbacks, the constant debugging—these are filters. They select for:

- **Patience:** The willingness to explain the same concept to an AI multiple times
- **Persistence:** The refusal to give up when the seventh attempted fix still doesn't work
- **Precision:** The discipline to verify every AI suggestion before accepting it
- **Adaptability:** The flexibility to change approaches when the AI can't follow your preferred path
- **Vision:** The ability to see past current limitations to future potential

These traits are not uniformly distributed. Many capable developers will dismiss vibe coding as "not ready yet" and return to traditional methods. They are not wrong—the tools *aren't* ready yet. But they will miss the window of learning, the accumulation of intuition, the development of AI-native thinking patterns.

The cave selects for those who see past what *is* to what *could be*.

VII. Towards Jarvis: What We're Building Toward

The Jarvis system in the Iron Man films represents the aspirational endpoint of AI-assisted development:

- **Complete Context Awareness:** Jarvis knows everything about every Iron Man suit, Tony's preferences, the current situation, and relevant history
- **Proactive Assistance:** Jarvis anticipates needs rather than merely responding to commands
- **Natural Dialogue:** Conversation flows naturally, with Jarvis understanding nuance, sarcasm, and implicit intent
- **Zero Hallucination:** Jarvis does not make things up; its information is reliable
- **Autonomous Execution:** Jarvis can be trusted to carry out complex tasks without micromanagement

We are not there yet. We may not be there for years. But the trajectory of AI development suggests we *will* get there, and when we do, the developers who learned to collaborate with primitive AI will become the architects of superhuman systems.

VIII. Conclusion: The First Suit

VibeSwap is my Mark I.

It is crude in places. The debugging sessions were painful. There are scars in the codebase where the AI and I fought and compromised. The modal state management in SwapCore.jsx went through four refactors before it stopped breaking. The device wallet flow has been fixed so many times that its history reads like a war journal. The personality quiz took three deploys to route correctly.

But it *works*. It trades. It bridges. It protects users from MEV. It runs.

And more importantly, in building it, I have developed intuitions and skills that will compound as AI tools improve. I have learned to communicate with machine intelligence, to verify its outputs, to correct its course, to persist through its limitations. I have built in a cave, with a box of scraps, and emerged with something functional.

The day will come—perhaps sooner than we think—when AI development assistants are so capable that solo developers can build systems that would today require teams of dozens. When debugging happens automatically, when context is infinite, when hallucinations are eliminated, when the AI truly understands your intent and executes it flawlessly.

On that day, those who dismissed vibe coding as "not ready" will scramble to learn what we already know. And those of us who built the first suits in caves will be ready to build the future.

Until then, we continue. We debug. We persist. We believe.

Because greatness can overcome limitation.

References

1. Stark, A. (2008). "Arc Reactor Miniaturization Under Resource Constraints." *Proceedings of Cave Engineering*, 1(1), 1-15.
2. Anthropic. (2024). "Claude: Constitutional AI and Capability Improvements." Technical Report.
3. OpenAI. (2023). "GPT-4 Technical Report." arXiv preprint.
4. Glynn, W. (2018). "Wallet Security Fundamentals: Your Keys, Your Bitcoin." Self-published whitepaper.
5. Various. (2025). "The Emerging Practice of Vibe Coding." Developer community discussions and blog posts.

"I am Iron Man."

— Tony Stark

"I am a vibe coder."

— Solo builders everywhere, 2025

Appendix: Lessons from the Cave

What Works

1. **Session state files** (CLAUDE.md, SESSION_STATE.md) dramatically improve cross-session continuity
2. **Explicit pattern documentation** reduces repeated mistakes

3. **Incremental changes** with immediate verification catch AI errors early
4. **Simple solutions** over clever solutions—the AI follows simplicity better
5. **Parallel tool calls** for independent operations maximize efficiency

What Doesn't Work

1. **Trusting without verifying**—always check AI output
2. **Complex refactors in single steps**—break into smaller changes
3. **Fighting the AI's limitations**—adapt your approach instead
4. **Assuming context retention**—re-state important information
5. **Skipping deployment verification**—always confirm changes work in production

The Meta-Lesson

The struggle is the curriculum. The frustration is the tuition. The debugging is the degree.

We are not just building software. We are building ourselves into the developers the future requires.

Document Version: 1.0 Written with Claude Code, in a cave, with a box of scraps.