ChatGPT

The Joy of Programming in the Age of AI-Assisted Development

by Dinis Cruz and ChatGPT Deep Research, 2025/07/03

Abstract

Abstract: This white paper explores how recent advances in generative AI are unlocking the *joy of programming* for a new wave of developers, often dubbed *"vibe coding"* practitioners. These individuals – many of them non-traditional programmers – are now able to create software through natural language and AI assistants, experiencing the same creative thrill and *flow state* that has long motivated career software engineers. We argue that AI-powered **no-code development** platforms dramatically lower the barrier to entry, enabling "citizen developers" in every business function to rapidly turn ideas into working applications. This democratization of programming is introducing millions of new people to the exhilaration of building and iterating software in real time. We examine key factors behind this phenomenon: the importance of immediate feedback loops (as emphasized by Bret Victor's *Inventing on Principle* [1] [2]), the intrinsic motivation developers feel when *"in the zone"* [3] [4], and how AI tools now deliver that instant interactivity to non-coders. While some fear that AI will obviate the need for human programmers, we highlight an opposite trend – a surge in programming participation and a greater need for professional developers to provide robust architectures, governance, and maintainability [5] [6]. In this thought-leadership piece, we present a vision of the near future where programming skills become widespread, every team includes AI-assisted developers, and the creative *joy* of coding becomes a universal experience rather than a specialist's privilege.

Introduction

In the software engineering community, the term *"flow state"* or being *"in the zone"* describes a peak experience of focus and creativity that makes programming deeply rewarding [3] [4]. A coder in this state loses track of time as they solve problems and see their ideas come to life, often working late into the night fueled by intrinsic motivation. This *joy of programming* – the magnetic allure of immersing oneself in code – is one of the core reasons many developers enter the profession [3]. For decades, however, this experience was largely limited to those who could overcome the high barriers of traditional software development: learning syntax, mastering complex toolchains, and navigating lengthy compile-test-debug cycles. Programming was a skill reserved for the relatively few, leaving most people as passive users of software.

Today, we stand at the threshold of a dramatic shift. Advances in generative AI and conversational coding assistants are *radically democratizing* software creation [7]. Natural language has effectively become the new programming language [7] – a high-level abstraction through which humans tell computers what to do. In this new paradigm (often informally called *"vibe coding"* or more professionally *No Code Development, NCD* [8]), a person can describe an application or feature in plain English and have an AI system generate the working code. Instead of painstakingly writing every line of code, the human can focus on *ideas* and *intent*, guiding the AI with prompts and examples. The machine handles the low-level details, while the human stays in the creative loop [9]. The end result is an **AI-assisted**

development process that feels fundamentally different – and immensely empowering – compared to traditional programming.

Crucially, this new mode of software development is introducing *a vast new audience* to the joy of programming. Business analysts, designers, project managers, and other professionals who once felt "locked out" of the software creation process can now actively participate in building tools and automations for their own needs. By removing the requirement to write code manually, generative AI is turning many "non-developers" into developers in practice [10]. These newcomers are beginning to experience the same creative highs and "aha" moments that career programmers cherish. They can brainstorm an idea, implement it through an AI assistant, instantly see it running, and iteratively refine it – all in a single day or even a few hours. This is a stark contrast to the old enterprise IT workflow where a businessperson's feature request might sit in an IT backlog for weeks. As we will discuss, the *immediacy* of this feedback loop is a game-changer: it creates a tight connection between **intent and outcome** that fuels engagement and innovation [1] [2].

In this paper, we explore the key ideas behind this phenomenon and its implications. We begin by examining the psychological payoff of programming – the state of flow and creative pleasure long noted in software development literature. We then show how AI-driven *"vibe coding"* tools recreate those conditions of rapid feedback and exploration for a broader population. Next, we discuss the emergence of **GenAI developers** (AI-assisted developers) and citizen development, highlighting industry trends that predict an explosion in the number of people creating software. Finally, we consider the broader impacts: why this movement will increase rather than decrease the need for traditional software engineering skills, and how organizations can harness this influx of new programmers while maintaining software quality and governance. Throughout, we maintain a forward-looking perspective, arguing that embracing this democratization of coding will unlock unprecedented creativity and productivity in the software field. The joy of programming, once the domain of the few, is poised to become a mainstream cultural experience – a development we should welcome and nurture.

The Joy of Programming: Flow State and Creative Delight

Software developers have often likened coding to a form of art or craftsmanship, noting the deep *satisfaction* and focus it can provide [11] [4]. Thomas De Moor aptly describes programming's "magnetic quality" – it *pulls you in*, scratching an itch in the mind that makes you want to keep solving problems [3]. When coding, hours can pass in what feels like minutes; a programmer may become so absorbed in the task that they lose awareness of time and external concerns [12] [13]. Psychologists call this optimal experience the **flow state**, and it is characterized by intense concentration, a sense of control and clarity, a distortion of temporal perception, and an intrinsic reward in the activity itself [14] [15]. In his seminal work, Mihály Csíkszentmihályi identified programming as one of many activities (like painting or sports) that can induce flow, because it provides clear goals, immediate feedback, and a balance of challenge and skill. In practice, developers often refer to this as *"being in the zone"* – a state where *"time passes in the blink of an eye, brilliant solutions appear, [and] it all just flows out automatically"* [13].

Crucially, the flow state is not just pleasurable – it also correlates with heightened productivity and creativity. A developer in flow can produce elegant solutions and leaps of insight that might not occur in a more distracted frame of mind [13]. This is one reason why developers cherish uninterrupted time (hence the classic image of coders wearing headphones to avoid interruptions). The flow experience is *autotelic*, meaning it is rewarding for its own sake; many programmers point to these moments of creative immersion as the reason they love their job, even beyond external rewards [14] [16]. As one article notes, *"Programming feels this way because you can immerse yourself so deeply into it that you forget*

about yourself…This is a highly rewarding experience. You're in the zone" [3]. In short, the *joy of programming* is real and well-documented – it's the thrill of solving a puzzle, the pride of creation, and the almost meditative focus that coding can induce.

However, reaching this joyous *zone* has traditionally required a significant level of programming skill and a conducive environment. A novice programmer often struggles with syntax errors, confusing toolchains, and long pauses to troubleshoot issues, all of which break flow. Likewise, in corporate settings, a developer's flow can be broken by meetings, context-switching, or waiting on slow build processes. Bret Victor, in his influential talk *"Inventing on Principle,"* argued that one key to unlocking creative flow is **immediate feedback**: *"Creators need an immediate connection to what they create…if you make a change, you need to see the effect of that immediately."* [1]. Traditional coding often imposes a delay (compile times, deployment steps, etc.) between making a change and seeing the result, whereas an ideal creative environment would tighten that loop to seconds or less. Victor's live demos showed that when feedback is instantaneous, it *"enables exploration, which then gives birth to ideas which would otherwise never see the light of day."* [2] In other words, quick feedback doesn't just save time – it actually *fosters creativity*, allowing the creator to play with ideas fluidly and stay in that blissful zone of discovery. This insight is pivotal in understanding why AI-assisted development is so exciting: it dramatically accelerates the feedback loop for many kinds of programming tasks, potentially making the flow state more accessible to more people.

From Backlogs to Real-Time Creation: Empowering a New Class of Developers

For much of the software industry's history, there has been a strict separation between "the builders" (software engineers) and "the idea people" or end users (often business professionals). A business domain expert who had a concept for a new tool or process improvement usually had to *hand off* their idea to a technical team. The typical enterprise workflow involved writing formal requirements, tickets in a backlog, sprint planning, and weeks or months before a working prototype emerged. This process was not only slow; it was creatively stifling for the idea originator. By the time a developer implemented the feature, the business user's context or needs might have changed, or the result might miss the mark – requiring yet another cycle of clarification and waiting. The lengthy turnaround eliminated any sense of *"flow"* for the non-technical participant; they were essentially observers, not creators. As Dinis Cruz has noted, the old model forced business users to be extremely explicit upfront and then wait on others to deliver, leading to frustrating delays and missed opportunities for rapid experimentation. It was, in many ways, the antithesis of the quick feedback and iteration that drive creative joy.

Now, consider how generative AI and no-code development tools are turning this situation on its head. In the emerging paradigm, a business user can become the builder, at least for many types of applications. Using natural language prompts and intuitive interfaces, non-engineers can *directly create* software to solve their immediate problems [10]. Instead of filing a ticket for IT, a marketing manager might use an AI app builder to create a custom dashboard or workflow automation by themselves, in a matter of hours. They can test it immediately, see what works or doesn't, and refine the idea on the fly. The feedback loop that used to span weeks is now compressed into perhaps minutes. This **real-time creation** capability unlocks the same iterative, exploratory process that professional developers enjoy in a good coding session – but for people with no formal coding background. The result is that the *"idea people"* are no longer on the outside looking in; they are in the driver's seat, experiencing the thrill of *making* something functional with their own hands (so to speak) and seeing instant results.

This shift is palpable in the stories coming from organizations that have embraced citizen development. Business staff often report a new sense of agency and excitement when they can automate a task or prototype an app *on their own*. They enter a playful mode of trying out ideas ("What if the form could do X? Let me just ask the AI to add that…"), akin to a programmer tweaking code to achieve a desired outcome. Frustrations naturally still occur – e.g. when the AI's output isn't quite right – but even those frustrations are part of a *motivating* cycle of trial and improvement, much like a traditional dev chasing a bug fix. The crucial point is that **immediate feedback** is now available to *everyone*. The business creator doesn't have to wait for a developer to come back next week with changes; the AI will attempt them right away. This creates a feeling of *"flow"* for the creator: they can maintain momentum and focus, continually refining their idea without long interruptions. In essence, we are witnessing the extension of Bret Victor's principle (immediate connection to what you create) beyond professional coders to a much broader population. When the latency between ideation and realization approaches zero, creativity explodes – people feel free to experiment, since they can quickly see the outcome and adjust course. It engenders a joyful, game-like interaction with technology: *"let's see what happens if I try this… oh interesting, now how about we add that…"*

Figure: The creative empowerment of non-traditional developers. With AI assistants handling the technical details, individuals from all backgrounds can focus on high-level ideas – experiencing the same immersive "flow" and satisfaction that career programmers do. They are, metaphorically, given a blank canvas *on which to realize their ideas, with instant feedback guiding their creative process.*

The term *"vibe coding,"* although slang, captures a slice of this experience – the notion of *"just vibing"* with an AI to build something, without strict formality. However, as Dinis Cruz and others have pointed out, the label *"vibe"* can be misleading [8]. What these new developers are doing is not magic or merely goofing around; it is a real engineering workflow, just highly abstracted. In this paper we'll use the term **AI-assisted no-code development** to emphasize that genuine development is occurring (with architecture, logic, and UIs being created), even if no one is manually typing source code. And importantly, the *joy* these creators feel is the same *joy of programming* that has always existed – the euphoria of making a machine do something new, of solving a problem through one's own creativity. The difference is that now that feeling is accessible to those who don't know C# or Python, expanding the ranks of people who can partake in it. This democratization of creation is analogous to how word processors allowed anyone to publish documents (not just professional typesetters), or how digital video tools let anyone try their hand at filmmaking. By lowering technical barriers, we let more people

focus on the creative act itself, which is inherently gratifying. AI is enabling a wider swath of humanity to *"think in code"* without writing code – and as a result, many are discovering that they *love* it.

Generative AI Developers: Millions of New Programmers

The rise of AI-assisted development is not only a feel-good story about creativity; it is also a major workforce and industry trend. We are on the cusp of a boom in the population of people who identify as software creators. Research firm Gartner projects that by 2025, *"citizen developers"* (employees building applications with low-code or no-code tools) will outnumber professional developers **4 to 1** in large enterprises [17]. This is a staggering ratio that underscores how widely the ability to create software is spreading beyond traditional IT. Microsoft has similarly noted the potential for hundreds of millions of new apps to be built by non-traditional developers in the coming years (one oft-cited figure is **450 million** new low/no-code applications in the next five years) as the demand for software far outstrips the supply of professional coders [18]. While earlier waves of this citizen developer movement were driven by visual programming tools and platforms like Excel or Salesforce, generative AI is supercharging the trend. Now *anyone* who can describe what they want in natural language can potentially develop a working piece of software. The programming knowledge once required – understanding specific APIs, syntax, and frameworks – is increasingly being handled by the AI translator. In Andrej Karpathy's words, *"Software 3.0 is programmed with prompts…the new 'computer' is an LLM, and the new 'programming language' is English…This marks a radical democratization of software creation."* [19]

This democratization means we will witness an influx of *millions of new "programmers"* — though they may not call themselves that — across all sectors of society. Every department in a company (marketing, HR, finance, etc.) can have its own DIY software builders who create custom solutions (forms, analytics, workflows, mini-apps) to optimize their work. Small business owners and solo entrepreneurs are using AI copilots to build their websites and online stores without hiring developers. Educators are using no-code AI tools to create learning apps tailored to their students. Even kids and teenagers with no coding training are starting to play with generative AI to create simple games or bots, essentially learning programming concepts without realizing it. In short, *software creation is becoming a universal competency*, much like document editing or slide presentation became ubiquitous with earlier generations of office software.

It's important to clarify that *quantity* does not equate to *proficiency*. Not every citizen developer will be crafting complex, robust systems – in fact, many of their creations will be simplistic or have rough edges. But the significance lies in the shift of mindset: people are beginning to see problems around them as *software-solvable*, and they have tools at their disposal to attempt solutions on their own. The latent demand for software is enormous (there are far more ideas for useful apps than there are developers to code them), and AI-driven development is a release valve for that pent-up demand [20] [21]. Workers no longer have to suffer with a manual process just because IT is busy; they can try to automate it themselves. This "self-service" approach to software means a lot more experimentation and innovation at the edges of organizations. Some experiments will fail, but some will succeed brilliantly, and even the failures provide learning and new iterations at a pace that old IT delivery models could not match.

Interestingly, early evidence suggests that this blossoming of new developers will actually **increase** the demand for experienced software engineers, not reduce it. Why? First, the more software is created everywhere, the more integration and maintenance work is generated – tasks often best handled by professionals. Second, when a prototype built by a citizen dev becomes popular or mission-critical, it usually needs hardening (security, scalability, refactoring), which calls for professional expertise. Third,

organizations will need experts to curate and govern the AI development tools, provide reusable components (APIs, templates), and set best practices to avoid chaos. Simon Wardley humorously noted that due to the "explosion of demand caused by vibe coding, most organizations will end up needing **more** software engineers" [22]. Dinis Cruz echoes that the *solid architecture and tech stacks* underneath these AI-generated apps will *"need more, not less Devs"* to build and maintain [5]. In essence, professional developers are needed to build the **platforms and guardrails** that enable safe and effective no-code development at scale. When every employee becomes a "developer," the role of the traditional developer shifts upward – focusing on frameworks, infrastructure, and high-level design that ensure all these citizen-built apps don't reinvent the wheel or create undue risk. It's a bit like how the spread of writing increased the need for teachers and librarians: more creators generate more work to support those creators.

The relationship between AI-assisted newcomers and seasoned developers is also collaborative. Far from being adversaries, they complement each other's strengths. The new GenAI-powered dev tools can handle routine coding and provide scaffolding, which even professional developers benefit from (it automates boilerplate and frees time for more complex logic). On the flip side, experienced developers become **mentors and editors** in this ecosystem – reviewing AI-generated code, tweaking prompts to get better outputs, and embedding their domain expertise into the AI's knowledge base. There is an *irony* in the current scenario: the people who can get the most out of AI coding tools are often those who know coding best. An experienced programmer can prompt ChatGPT or a code generator in a very precise way, foresee pitfalls, and iteratively guide it to a solution much faster than a novice could [23]. Instead of making their skills redundant, the AI becomes a force multiplier for their productivity. Meanwhile, those with less coding skill can achieve results previously out of reach by leaning on the AI and perhaps consulting an expert for tough parts. The net effect is an *overall increase* in software output and a broadening of participation, with skilled engineers still very much in demand.

Quality, Craft, and the Evolving Role of Professionals

With so many new hands joining the proverbial keyboard (or rather, the prompt dialog), a natural question arises: what about software quality, security, and all the *non-functional requirements* that professional developers obsess over? Indeed, one of the key challenges in this new era is ensuring that the exuberance of rapid no-code development doesn't lead to a tangled mess of fragile applications or security nightmares. Early experiences with AI-generated code show a double-edged sword: the AI can produce functional code quickly, but it may also introduce hidden bugs, architectural anti-patterns, or "technical debt" if used naively [24] [6]. A citizen developer, for instance, might build a workflow that works for 10 users in a test, but fails at 1,000 users or exposes sensitive data inadvertently. The responsibility of catching and correcting such issues often falls to the professional developers or IT departments downstream. In other words, the *craft* of software engineering – writing efficient algorithms, securing data, ensuring maintainability, etc. – remains as vital as ever, and arguably becomes even more important when software is being produced by people unfamiliar with these concepts.

Organizations that successfully embrace AI-assisted development tend to do so with a *"fusion team"* approach: blending business domain experts with professional developers in cross-functional teams [25] [20]. The idea is to combine the creativity and domain knowledge of citizen devs with the technical rigor of seasoned devs. The AI tools serve as the bridge between them. For example, a business expert can draft a workflow using a no-code tool, and an IT expert can then review the generated artifacts, adding error handling, security checks, and performance tweaks as needed. Over time, some of these quality guardrails can be automated – e.g. the AI itself can be trained to enforce certain secure coding practices – but human oversight is still critical. A Medium article by Thoughtworks on generative AI in coding noted that while AI can speed up development, *"AI-generated code can compound technical debt if not*

properly managed," underscoring the necessity of human review and good software engineering practices to maintain quality [6]. In a sense, we are moving toward a model where AI handles the first draft of code, and humans act as editors and architects. The *first draft* comes quickly – capturing the intent – and then experts refine it to meet production-grade standards.

Professional developers, therefore, are not becoming obsolete; their role is evolving. They are the ones who will build the robust **building blocks** (libraries, APIs, services) that citizen developers snap together. They will also focus more on defining **policies and standards** (for security, data governance, UI/UX consistency, etc.) that the AI tools can be configured to follow. In many organizations, developers are already creating internal "prompt libraries" or templates for common tasks, which less technical staff can invoke without starting from scratch each time. Moreover, the traditional developers are themselves benefiting from higher-level abstractions – they too can code at a higher level with AI, which means they can attack more ambitious problems. Imagine one expert developer overseeing 10 citizen developers each building departmental apps: the expert might set up the core data models and APIs for all apps, while the citizen devs focus on their specific features. The result is faster delivery and a system that still adheres to a sound architecture defined by the expert. This kind of **hybrid teamwork** will likely become a norm. It resembles an apprentice model, except the "apprentices" are partly AI tools and partly human non-experts, all guided by master developers.

Another aspect to consider is the continuous improvement of the AI tools themselves. As generative models become more advanced, they will better handle non-functional requirements on their own (for instance, automatically suggesting performance optimizations or flagging potential security issues). Yet, even those capabilities often originate from human knowledge encoded into the models. The collective wisdom of the developer community – the decades of lessons about how to write good software – needs to be fed into these AI systems. That happens via training data, fine-tuning on high-quality code, or explicit rules integrated into AI coding assistants. Seasoned engineers play a key part in this feedback loop: by using AI tools and correcting their mistakes, they teach the AI what is acceptable or not. In the big picture, we can envision a future where much routine coding is handled by AI, but the overall system design and the creative problem-solving (as well as the accountability for the end product) remains with humans. Software engineering might shift to be less about typing syntax and more about **choosing the right abstractions, validating solutions, and steering AI** – all higher-level tasks that require experience and holistic thinking.

Finally, it's worth highlighting that the *joy of programming* we celebrated earlier is not diminished by these changes – if anything, it's amplified and spread to more participants. Professional developers may find that offloading grunt work to AI lets them spend more time in the enjoyable parts of development: designing, innovating, polishing, and learning new things. Meanwhile, the newcomers are discovering joy in what *before* might have been an intimidating or frustrating activity. The key is keeping that joy sustainable by avoiding disillusionment. That means ensuring these citizen devs have pathways to learn and improve (so they don't hit walls where they can't progress) and that their successful prototypes are nurtured rather than left to collapse under scaling issues. By investing in training, community, and support around AI development tools, organizations can cultivate a thriving maker culture. When someone without a coding background creates their first useful app and sees colleagues actually using it, the sense of accomplishment is huge – akin to a developer shipping their first product. By facilitating many such experiences, we not only make individuals happier and more empowered, but we also drive the organization forward through bottom-up innovation.

Conclusion

We are entering an era in which the act of programming – turning ideas into executable software – is no longer the exclusive domain of software engineers. Generative AI has given rise to a new cohort of *AI-assisted developers*, from hobbyists to business users, who can build applications through natural language conversations and intuitive interfaces. This transformation is bringing the *joy of programming* to the masses. Millions of people will taste that special satisfaction of seeing "Hello World" (or its equivalent) appear from something they created, then rapidly evolve that creation to be ever more useful. They will know the flow of staying up late to tweak a project not because they have to, but because they *want* to – driven by curiosity and passion. They will also inevitably feel the frustrations of debugging and the stubbornness of computers – but even those struggles can be rewarding when overcome. In short, coding is becoming a more human-centric, creative endeavor for all, rather than a technical skill for the few.

This white paper has outlined how and why this is happening: AI short-circuits the feedback loop and removes the requirement of formal coding knowledge, thus enabling immediate creative expression in software. We saw that immediate feedback is a catalyst for flow and innovation [1] [2] – a principle now being applied on a broad scale. We also discussed the ramifications for the software industry and organizations: a surge of new developers, the need for guiding frameworks, and the evolving role of professional engineers. Rather than fearing a loss of control or quality, forward-thinking teams recognize this wave of citizen development as an opportunity. They are putting in place the tools, training, and governance to support it. Those who succeed will effectively multiply their development capacity manyfold, as every knowledgeable worker becomes also a software creator to some degree. As Gartner's analysis suggests, the majority of software in the near future could be written by folks outside of traditional IT [17] – a profound shift that can drive unprecedented agility and innovation if harnessed well.

In embracing this change, it's important to keep sight of the human aspect at its core: **joy and empowerment**. The technologies involved – large language models, low-code platforms, etc. – are means to an end. The end is empowering people to solve their own problems and explore their ideas with less friction. It is fitting to recall that the early personal computing revolution was driven by a similar ethos: the PC spread computing power to the masses, enabling a generation of creators. Today's AI tools are doing the same for software development itself. When a young entrepreneur or a schoolteacher or a scientist can *develop* an app just by conversing with an AI, we have essentially given them a superpower that was previously limited to those with years of coding education. What great things might come from unleashing this creative potential broadly? We can expect an outpouring of niche solutions, personalized tools, and community-driven software that would never have existed under the old model.

To be clear, the road ahead will have challenges. We will need to educate this new population of developers about principles of good software (much as we teach digital literacy). We will need robust AI ethics and safety to ensure the tools guide users toward sound outcomes. We will likely redefine job roles and team structures to integrate AI-developed components. But the momentum is undeniable: programming is becoming more conversational, more visual, and more accessible. The cultural narrative is already shifting from *"learn to code or you won't understand the future"* to *"you don't need to write code to create software"*. This is a paradigm shift that can widen the tent of technology creators dramatically. And as that tent widens, the collective joy and creativity in our industry grows with it. The essence of programming – logical thinking, problem solving, creativity – remains, but the entry path is welcoming to many more.

In conclusion, the *joy of programming* is evolving into a joy of *creating* with AI as an ever-present partner. It is a joy shared by veteran programmers and newcomers alike, each in their own way collaborating with intelligent tools to build something meaningful. We are optimistic about this future. Imagine a world where a marketing specialist, a nurse, or an artist can all harness the power of software to bring their ideas to life without needing a computer science degree – that world is emerging now. The role of leaders and innovators is to guide it responsibly, ensuring that this revolution yields durable benefits. If we do so, we will see technology truly become a universal language of innovation. As one tech visionary quipped, the new programming language is English [7] – but beyond the humor of that statement lies an inspiring reality: *anyone* can now speak the language of creation with computers. And the feeling of *"it works!"* – that little spark of joy when your program runs as intended – is something everyone deserves to experience. By embracing AI-assisted development, we make that possible, lighting the path for a new generation of creators to join the party and revel in the joy that comes from coding something awesome.

References:

1. De Moor, T. (2020). *How to Fall Into a Flow State of Mind*. X-Team. – Describes why programming has a magnetic quality and how being "in the zone" (flow state) makes developers feel happy and deeply focused [3] [14].

2. Impact Lab (2013). *Why software developers are like artists*. – Draws parallels between coding and art, noting that both involve getting "in the zone" where time flies and creativity flows [4].

3. Victor, B. (2012). *Inventing on Principle* (Talk). – Introduces the principle that creators need an immediate connection to what they create, showing that instant feedback enables new ideas and maintains creative flow [1] [2].

4. Cruz, D. (2025). *No Code Development (NCD): A Paradigm Shift Beyond 'Vibe Coding'*. – White paper by Dinis Cruz and ChatGPT Deep Research arguing for the term NCD. It discusses how in AI-assisted development the human stays in creative flow while the AI handles code generation [9], and notes that experienced engineers are not obsolete but often excel at this new paradigm [23].

5. LinkedIn Post by Dinis Cruz (2025). *"Vibe coding: a bad name for a good idea"*. – Commentary suggesting that "vibe coding" (AI-based coding) adds value by letting users create custom UIs on solid tech stacks, which will require **more** developers, not fewer [5]. Emphasizes that the term "vibe" is misleading and it's really about user-driven coding with AI.

6. LinkedIn Comment by Simon Wardley (2025) – Notes five points about vibe coding, including that prompts are not deterministic commands, LLMs always hallucinate to some degree, and critically that *due to the explosion of demand from vibe coding, most organizations will need more software engineers* (not fewer) [22].

7. Thoughtworks (May 2025). *Can vibe coding produce production-grade software?* – A Medium article analyzing AI-assisted development. It found that AI can generate working code quickly but struggles with complex changes, requiring human intervention. It highlights that less-technical team members can effectively prototype ideas with these tools, lowering the barrier to entry [10], but cautions about technical debt and the need for human oversight to ensure maintainable code [6].

8. Gartner via Kissflow (2025). *Rise of the Citizen Developer* – Reports Gartner's forecast that by 2025, citizen developers in large enterprises will outnumber professional developers 4:1 [17]. Explains why citizen development is taking off (overloaded IT, need for agility) and how no-code tools let non-technical users build solutions rapidly.

9. Karpathy, A. (2025). *Software 3.0 and Vibe Coders* (as covered by S. Ranjula, Verdentra) – Describes Andrej Karpathy's vision that *"the new programming language is English"* and that software is now built with prompts to LLMs [7]. Indicates a huge amount of software will be written by a *"new generation of 'vibe coders'"* who are not traditional engineers [26], marking a democratization of coding.

10. Medium (2017). *Coding Has Become Pop Culture* (Attila Vágó) – Critiques the oversimplification of coding for the masses but notes the push to create *"millions of new programmers by 202x"* through educational tools [18]. (Included for context on the long-running narrative of making coding accessible, which AI is now accelerating).

---

[1] [2] Immediate Feedback in Programming
https://www.reillywood.com/blog/inventing-on-principle/

[3] [12] [14] [15] [16] How to Fall Into a Flow State of Mind | X-Team
https://x-team.com/magazine/flow-state-of-mind

[4] [11] [13] Why software developers are like artists – Impact Lab
https://www.impactlab.com/2013/10/24/why-software-developers-are-like-artists/

[5] [22] spot on analysis on Vibe coding | Dinis Cruz
https://www.linkedin.com/posts/diniscruz_spot-on-analysis-on-vibe-coding-the-area-activity-7329090329624653825-kT5h

[6] [10] [24] Can vibe coding produce production-grade software? | by Thoughtworks | May, 2025 | Medium
https://thoughtworks.medium.com/https-www-thoughtworks-com-insights-blog-generative-ai-can-vibe-coding-produce-production-grade-software-75130f25b63d

[7] [19] [26] From Code To Vibes: Andrej Karpathy On The New Era Of Software | Verdentra
https://www.verdentra.com/article/from-code-to-vibes-andrej-karpathy-on-the-new-era-of-software/

[8] [9] [23] No Code Development (NCD): A Paradigm Shift Beyond 'Vibe Coding' - Dinis Cruz - Research Hub
https://docs.diniscruz.ai/2025/06/18/no-code-development--ndc--a-paradigm-shift-beyond-vibe-coding.html

[17] [20] [21] [25] What Gartner Says About the Rise of the Citizen Developer | Kissflow
https://kissflow.com/citizen-development/gartner-on-citizen-development/

[18] Coding Has Become Pop Culture. But programming has not. And let me... | by Attila Vágó | Bricks n' Brackets | Medium
https://medium.com/bricksnbrackets/coding-has-become-a-pop-culture-939100f84b0c