www.ijecs.in
International Journal Of Engineering And Computer Science
Volume 14 Issue 11 November 2025, Page No. 27813-27818
ISSN: 2319-7242 DOI: 10.18535/ijecs/v14i11.5292

Democracy in Software Development: The Rise of Vibe Coding

Abubakar Bello Bada, Chizzy Ifesinachi Ede, Ibrahim Musa Mungadi, Mubashir Haruna
Computer Science Department, Federal University Birnin-Kebbi, Kebbi State - Nigeria

Abstract
Software development as an engineering discipline is characterized by tension between abstraction and precision. It has undergone a tremendous transformation over the decades, from highly rigid machine language programming to the modern day *vibe coding* that tends to democratize software development through automation, abstraction, and artificial intelligence (AI). Vibe coding, a term that refers to AI-assisted and intuition-driven software development methodology. This paper first provides the historical trajectory of software development, arguing that each stage has incrementally democratized software development. The current shift powered by Large Language Models (LLMs) represents the most significant stride in the democratization of software development yet. This paper also enumerates the implications of this shift and the evolution of software development expertise. It concludes that while vibe coding has its challenges, it aligns with the historical evolution of software development, which is the relentless pursuit of higher-level abstraction to harness human creativity and collective intelligence.

1. Introduction
The history of software development or programming is essentially, a history of creating ever more sophisticated tools to manage complexity and bridge the gap between human intent and machine execution. The path of software development reflects an unending pursuit of accessibility, efficiency, and expressiveness (Marar, 2024). From the machine language of the 1940s to the emergence of transformer-based large language models and AI-assisted platforms of today, each transformation or paradigm shift has pushed software development closer to a democratic process where more individuals can participate. The rise of *vibe coding*, which is a practice where developers express a high-level goal – "vibe" – through natural languages and an AI agent generates the requisite low-level code to realize that intent, represents the achievement of this historical trend (Maes, 2025; Sarkar & Drosos, 2025).
This paper maintains that *vibe coding* is not a digression but a progression in the long-standing trend of democratization of software development. To prove this claim, this paper first present a historical background of software development paradigms. It then conceptualizes *vibe coding*, and discusses its implications for the future of software development.

2. Historical Evolution of Software Development
Here is a historical analysis of software development.
2.1 Machine Language (1940s – early 1950s)
The first programming language to emerge is machine language. This language comprised of binary codes (0s and 1s) that allows programmers to give direct commands to computer hardware (Komil, 2025). This programming, while efficient, is error-prone, tedious, and limited to experts with deep knowledge of computer architecture (Rane, 2023). This complexity made computing in that era an elite practice. Despite the challenges, machine language was crucial in creating foundational programs like IBM 701 and others.

2.2 Assembly Language (1950s – 1960s)
Assembly language follows immediately after the era of machine language. When programmers thought of ways to make programming more accessible, assembly language was invented. It is the human-readable

representation of machine language, which provide mnemonic codes and symbolic names to represent machine instructions (Ibrahim et al., 2025).
The introduction of mnemonics and symbolic representations that came with assembly language slightly improved readability of programming while retaining hardware specificity (Harris & Harris, 2022). Although assembly language is still complex, it enabled more programmers to engage with computer thereby laying the foundation for systematic programming practices.

2.3 High-Level Languages (1960s – 1980s)

The introduction of High-level languages saw the development of compiled languages such as FORTRAN, COBOL, BASIC, and later C; this hid low-level details like memory management and hardware interactions and introduced abstraction layers such as loop, variables, and functions (Kudebayeva, 2025). High-level languages enabled programmers to express algorithms using syntax closer to mathematical notation or English, while a compiler handled the tedious and error-prone translation to machine code (Komil, 2025). This era marked an important step in democratization of software development as programming became accessible to many people.

2.4 Structured and Object-Oriented Programming (OOP) (1970s – 1990s)

Between 1970s and 1990s, structured and Object-Oriented programming emerged in response to challenges of creating large complex software. Structured programming, which gained prominence in the 1970s and 1980s was a response to source code that are poorly structured, complex, and difficult to understand and maintain (Rane, 2023). Structured programming focused on creating clear and linear flow of control through a limited set of control structures thereby creating code that is easier to read and debug. OOP, which gained prominence in 1980s and 1990s, on the other hand took a different approach. It modeled software around objects and behavior (Azizbek, 2024).
While structured programming focused on functions that operate on data, OOP organized programs around objects that contain both data and the methods to manipulate it, leading to more modular, reusable, and maintainable code, which became essential for handling the growing complexity of modern software (Benjamin, 2024). With structured and Object-Oriented programming, software design became more scalable and maintainable, enabling large-scale industrial software development.

2.5 Integrated Development Environments (IDE) and Agile (1990s – 2000s)

The 1990s and 2000s saw the transformation of software development by IDEs and agile methodologies. IDEs, like Eclipse and Visual Studio simplify coding by combining code editors, debuggers, and build automation tools into a single application (Chacón et al., 2022). This centralizes the whole software development processes, which in turn boost productivity. In 2001, agile manifesto formalized the agile methodologies in response to the rigid waterfall model. Agile methodology emphasized iterative, collaborative development, and adapting to changing requirements through collaboration between development team and their customers: this, therefore, shift software development from individual expertise to teamwork (Arrasyid et al., 2024; Chacón et al., 2022). These two innovations revolutionized software development by making it more efficient, collaborative, and responsive.

2.6 Open Source Movement (1990s – present)

In the 1990s, a philosophy and social movement, Open Source movement, which champions the free availability of software source code, emerged. This philosophy advocates for a development model where anyone can view, modify, and distribute code (Curto-Millet & Corsín, 2023). This collaborative approach initiated by projects like Linux helped in bringing innovation and serves as a foundational element of the internet and modern software (Li et al., 2025). Platforms like GitHub and GitLab made source code widely accessible, reinforcing democratic ideals by allowing contributions from diverse developers worldwide.

2.7 Cloud Computing and DevOps (2010s)

In the 2010s, cloud computing and DevOps emerged as a transformative pairing, with cloud platforms providing scalable, pay-as-you-go infrastructure for DevOps practices like Continuous Integration/Continuous Deployment (CI/CD) and automation (Datla, 2024). This synergy revolutionized

software delivery and allowed developers and operations teams to deliver applications more quickly and reliably, and improve collaboration (Brooks et al., 2025; Das Chowdary et al., 2024).
Cloud Computing and DevOps democratized software development by making infrastructure and tools more accessible, reducing costs, enabling faster iteration, and fostering a more collaborative and automated approach to building and delivering software.

2.8 AI and Low-Code/No-Code (2020s – present)

AI is the technology that enables computers perform intelligent tasks without explicitly been programmed while Low-Code/No-Code (LCNC) are platforms that use visual interfaces for software development without the need for extensive coding (Ilesanmi, 2025). LCNC enables individuals with little to no coding experience to build applications as they use graphical interface, drag-and-drop features, and pre-built integration to simplify software development (Gaurika, 2024). With the advent of AI, these platforms become more enhanced.
Generative AI models (e.g., GPT, Codex) and low-code platforms expanded programming access to non-specialists (Ren et al., 2023). Developers now collaborate with AI agents thereby blurring the lines between coding expertise and creative problem-solving.

3. The Rise of Vibe Coding

The current paradigm shift made possible by LLMs built upon all previous abstractions, however, it represents a qualitative and progressive change in human-computer interaction.

3.1 Conceptualizing Vibe Coding

Vibe coding refers to an informal description of software development method where a developer uses natural language in form of a prompt to express his/her intent while an AI agent (GenAI LLM) generates the resultant code (Maes, 2025). Unlike traditional programming which is defined by strict syntax and planning, vibe coding thrives on iterative exploration. The "vibe" in vibe coding refers to imprecise, high-level description of the desired outcome (Sarkar & Drosos, 2025).

3.2 Comparison between Vibe Coding and Traditional Coding

Here is a breakdown of some of the differences between vibe coding and traditional coding.

	Vibe coding	**Traditional coding**
Syntax dependency	Low	High
Creativity	Enhanced through collaboration with AI	Limited by formalism
Accessibility	Wider populace including non-programmers	Skilled programmers only
Interaction method	Conversational using natural language	Writing codes directly line-by-line
Focus	Idea shaping	Problem-solving guided by rules
Code understanding	Full understanding of the code not required	Absolute understanding as the developer is the sole author and must have full understanding of what he/she authored
Iteration speed	Moderate	Extremely fast

4. Implications of Vibe Coding for Software Development

4.1 Democratization of Programming

Just like the invention of personal computers and the advent of visual programming tools, vibe coding lowers the entry barrier for non-technical populace to engage in software development (Sarkar & Drosos, 2025). By lowering the entry barrier, vibe coding has enabled a new class of "citizen developers" to create software without formal programming training. The democratization of software development enabled by vibe coding enables domain experts e.g., financial analyst, with little to no coding experience to generate

functional scripts in order to solve their specific problems by describing them in natural language, thus blurring the lines between users and developers (Marko, 2024).
Unlike the democratization brought by previous paradigm shifts, from machine language to high-level languages, this democratization has a broader reach.

4.2 Redefinition of the Programmer's Role

With the advent of vibe coding, the role of programmers or software developers will not become obsolete but will transform. The role will transform from the ability to write codes to ability to create effective prompts (prompt engineering), designing the overall structure of the system, critically auditing AI-generated codes, selecting the right AI-generated output and integrating it seamlessly into a larger system, and solving novel problems (Akhilesh Gadde, 2025; Gadde, 2025).

4.3 Acceleration of Innovation

Vibe coding accelerates innovation by enabling developers to quickly build and test ideas, shifting the developers' focus from technical implementation to creativity and user experience. With AI handling repetitive tasks, developers can rapidly prototype, test, and refine systems (Crowson et al., 2025). This acceleration supports innovation in domains where traditional coding bottlenecks limited experimentation.

4.4 Challenges and Risks

The rise of vibe coding paradigm though a step in the right direction, it has its own challenges and risks (Gadde, 2025; Maes, 2025; Marko, 2024). The following are some of them:

i. Over-Reliance on AI: over-reliance on AI by software developers for software development tasks can lead to skill erosion among developers. This skill erosion will lead to a generation of developers without good understanding of data structures, algorithms, and memory management, all of which are the very fundamentals that allow for critical evaluation of AI output.
ii. Bias and Hallucinations: LLM though statistically sophisticated, lacked true understanding. They can be biased towards a certain programming strategy or generate codes that are incorrect or insecure. Blind acceptance of these codes introduces "Hallucination debt", a new form of technical debt caused by unverified AI-generated results.
iii. Security Concerns: since LLM are mostly trained on public codes, they may replicate vulnerabilities in their training data. This can become a powerful tool for attackers to create a novel malware or use it to find exploits.
iv. The Homogenization of Code: If the majority of developers use the same few AI models, there is a risk that codebases will become homogenized, potentially reducing diversity of thought and innovation in problem-solving approaches.
v. Intellectual Property Issues: generating codes by AI has the potential of violating open source license and infringing copyright because generated code may replicate protected materials. This is a big issue because the ownership of the generated codes remains an issue, as the legal landscape surrounding code generated by models trained on copyrighted open-source code is still undefined.

4.5 Educational Implications

The implication of vibe coding in education is enormous most especially programming. It shifts focus from implementation to conceptual understanding and creativity (Chow & Ng, 2025; Crowson et al., 2025). Curricula must shift toward teaching problem decomposition, system thinking, and AI literacy rather than routine programming syntax.

4.6 Ethical and Socio-Economic Implications

The evolution of vibe coding, though revolutionary, raises important ethical and socio-economic considerations that merit close attention within the computing and AI research community. From an ethical standpoint, vibe coding challenges the boundaries of authorship, responsibility, and accountability in software creation. When AI agents generate code, questions arise regarding who is responsible for potential errors, security vulnerabilities, or unethical applications of such software (Floridi & Cowls, 2022). Without

clear accountability frameworks, vibe coding risks creating a "responsibility gap" where neither the developer nor the AI provider assumes full ownership of outcomes.

From a socio-economic perspective, vibe coding could reshape the global software labor market. While it democratizes access to coding by enabling non-experts to participate, it may also displace junior or entry-level programmers whose work often involves repetitive or routine coding tasks (Nieborg & Poell, 2023). In developing nations, where software outsourcing forms a vital part of the digital economy, this disruption could exacerbate existing economic inequalities and reduce employment opportunities.

Moreover, access to AI-powered coding tools is still largely restricted to individuals or institutions in technologically advanced regions. This uneven access risks amplifying what scholars have termed "AI colonialism," where innovation and control of AI infrastructure are concentrated in a few dominant economies (Mohamed, Png, & Isaac, 2020). To ensure true democratization, global efforts must focus on making AI-assisted coding resources open, affordable, and inclusive.

5. Conclusion

The history of software development reveals a progressive movement toward democratization: from exclusive binary coding to inclusive AI-driven creativity called "vibe coding". Vibe coding represents the latest and the most disruptive stage of this evolution, where the synergy between human intuition and machine intelligence transforms how software is conceived, designed, and built. While this shift introduces great challenges e.g. reliability, ethics, and security, vibe coding holds the promise of reimagining software engineering as a participatory, creative, and democratic practice.

AI alone will not write the future of software development, but the symbiotic partnership between human intuition and vision on one hand and artificial intelligence on the other will. The challenge for the future is to build tools and educational frameworks that will take care of challenges brought by this paradigm. Security, bias, ethics, and validation need tools that software developers will use in ensuring they overcome them, while skills shift needs educational frameworks in order to train, re-train, and upskill developers. These will make sure that vibe coding not only significantly lowers the barrier for entry into software development, it also increases the productivity of the existing developers.

References

1. Akhilesh Gadde. (2025). Democratizing Software Engineering through Generative AI and Vibe Coding: The Evolution of No-Code Development. *Journal of Computer Science and Technology Studies*, *7*(4), 556–572. https://doi.org/10.32996/JCSTS.2025.7.4.66
2. Azizbek, R. R. (2024). *REPLACE OBJECT ORIENTED PROGRAMMING (OOP) IN PYTHON PROGRAMMING LANGUAGE*. https://universalpublishings.com
3. Benjamin, C. P. (2024). *Advanced Topics in Types and Programming Languages*. https://books.google.com.ng/books?hl=en&lr=&id=YsEIEQAAQBAJ&oi=fnd&pg=PR9&dq=structured+programming+languages&ots=-e08zhtU5e&sig=kJfoUOiCOVyau7Hr-Fun96Cw-ew&redir_esc=y#v=onepage&q=structured%20programming%20languages&f=false
4. Brooks, N., Vance, C., & Ames, D. (2025). Cloud Computing: A Review of Evolution, Challenges, and Emerging Trends. *Journal of Computer Science and Software Applications*, *5*(4). https://doi.org/10.5281/ZENODO.15165068
5. Chacón, J., Besada-Portas, E., Lía García-Pérez, ·, Jose, ·, & López-Orozco, A. (2022). *An integrated framework for the agile development and deployment of low cost remote laboratories*. https://doi.org/10.1007/s11042-024-20306-8
6. Chow, M., & Ng, O. (2025). From technology adopters to creators: Leveraging AI-assisted vibe coding to transform clinical teaching and learning. *Medical Teacher*. https://doi.org/10.1080/0142159X.2025.2488353
7. Crowson, M. G., Mbi, M., Celi, L., Celi, A., & Mpa, M. D. (2025). *Academic Vibe Coding: Opportunities for Accelerating Research in an Era of Resource Constraint*.
8. Curto-Millet, D., & Corsín, J. A. (2023). The sustainability of open source commons. *European Journal of Information Systems*, *32*(5), 763–781. https://doi.org/10.1080/0960085X.2022.2046516/ASSET/EA83B81C-1F5E-42B5-863F-7AD8B62FE625/ASSETS/GRAPHIC/TJIS_A_2046516_F0002_OC.JPG

9. Das Chowdary, V. H., Shanmukh, A., Nikhil, T. P., Kumar, B. S., & Khan, F. (2024). DevOps 2.0: Embracing AI/ML, Cloud-Native Development, and a Culture of Continuous Transformation. *Proceedings - 2024 4th International Conference on Pervasive Computing and Social Networking, ICPCSN 2024*, 673–679. https://doi.org/10.1109/ICPCSN62568.2024.00112
10. Datla, V. (2024). The Evolution of DevOps in the Cloud Era. *Journal of Computer Engineering and Technology (JCET)*, *6*(1), 7–12. https://iaeme.com/Home/journal/JCET7editor@iaeme.comAvailableonlineathttps://iaeme.com/Home/issue/JCET?Volume=6&Issue=1
11. Gadde, A. (2025). *Democratizing Software Engineering through Generative AI and Vibe Coding: The Evolution of No-Code Development*. https://doi.org/10.32996/jcsts
12. Gaurika, S. (2024). *The Role of AI in Low-Code and No-Code Platforms*. https://medium.com/accredian/the-role-of-ai-in-low-code-and-no-code-platforms-c801169390a3
13. Harris, S. L., & Harris, D. (2022). Architecture. *Digital Design and Computer Architecture*, 298–390. https://doi.org/10.1016/B978-0-12-820064-3.00006-4
14. Ibrahim, M., Ibrahim, M., Ibrahim, A., Nura, T., Maigari, A., & Umar, M. (2025). *Assembly Programming: An In-Depth Analysis and Applications Corresponding Author **. https://doi.org/10.35248/IJIRSET.24.05(2).002
15. Ilesanmi, M. (2025). *Introduction to No-Code AI Platforms and Their Role in AI Democratization*. https://www.researchgate.net/publication/392501733_Introduction_to_No-Code_AI_Platforms_and_Their_Role_in_AI_Democratization
16. Komil, T. J. (2025). *PROGRAMMING LANGUAGES AND ITS TYPES*. https://doi.org/10.5281/zenodo.15447946
17. Kudebayeva, A. (2025). *Towards Learnable Programming: The Role of Abstraction and Modularity in Interactive Spreadsheet Programming*. https://qmro.qmul.ac.uk/xmlui/handle/123456789/106251
18. Li, B., Liu, C., Fan, L., Chen, S., Zhang, Z., & Liu, Z. (2025). Open Source, Hidden Costs: A Systematic Literature Review on OSS License Management. *IEEE Transactions on Software Engineering*. https://doi.org/10.1109/TSE.2025.3586411
19. Maes, S. H. (2025). *The Gotchas of AI Coding and Vibe Coding. It's All About Support And Maintenance*. https://doi.org/10.5281/zenodo.15343349
20. Marar, H. W. (2024). Advancements in software engineering using AI. *Computer Software and Media Applications*, *6*(1), 3906. https://doi.org/10.24294/csma.v6i1.3906
21. Marko, H. (2024). *What is Vibe coding and when should you use it (or not)?* https://www.researchgate.net/publication/394313040_What_is_Vibe_coding_and_when_should_you_use_it_or_not
22. Arrasyid, R., Teguh, R., & Ni Wayan Trisnawaty. (2024). Integration of User Experience and Agile Software Development: A Systematic Literature Review. *The Indonesian Journal of Computer Science*, *13*(6). https://doi.org/10.33022/IJCS.V13I6.4466
23. Rane, P. P. (2023). The Evolution of Computer Programming Languages. *International Journal of Advanced Research in Science, Communication and Technology (IJARSCT) International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal*, *3*(1). https://doi.org/10.48175/IJARSCT-13110
24. Ren, X., Ye, X., Zhao, D., Xing, Z., & Yang, X. (2023). *From Misuse to Mastery: Enhancing Code Generation with Knowledge-Driven AI Chaining*. http://arxiv.org/abs/2309.15606
25. Sarkar, A., & Drosos, I. (2025). *Vibe coding: programming through conversation with artificial intelligence*. http://arxiv.org/abs/2506.23253
26. Floridi, L., & Cowls, J. (2022). *A Unified Framework of Five Principles for AI in Society*. Harvard Data Science Review, 4(1). https://doi.org/10.1162/99608f92.bb6b4d4a
27. Nieborg, D. B., & Poell, T. (2023). *The Platformization of Cultural Production*. Cambridge, MA: Polity Press.
28. Mohamed, S., Png, M., & Isaac, W. (2020). *Decolonial AI: Decolonial Theory as Sociotechnical Foresight in Artificial Intelligence*. Philosophy & Technology, 33(4), 659–684. https://doi.org/10.1007/s13347-020-00405-8