Exploring the Potential of LLMs in the SDLC: A Productivity Perspective

Tech Report

Introduction

In a software industry that thrives on faster iteration, collaborative efficiency, and measurable value, Large Language Models (LLMs) are moving from the sidelines into the heart of the Software Development Life Cycle (SDLC). The session blended research, real-world insight, and audience engagement to explore how LLMs can enhance developer productivity [1].

Even though 'vibe coding' seems to be gaining traction, there's still a lot of uncertainty about whether this is the best path forward [2]. Many believe that LLMs and AI tools should serve as enablers, helping developers, not replacing sound software engineering practices.

Current research shows that LLMs represent a major advancement in automating and enhancing key stages of the software development lifecycle, offering the potential for increased productivity and higher-quality software [3-5]. At the same time, their effective integration into the SDLC still faces notable challenges. Concerns such as data privacy, the need for domain-specific training data, architectural and contextual limitations, and the lack of robust metrics to evaluate both LLM performance and developer productivity remain open areas for investigation. Furthermore, assessing the impact of LLMs is particularly complex, as existing evaluation methods may not fully capture the breadth of their contributions.

This document presents an overview of the talk [1]; the results gathered from the audience and some final considerations and future directions to proceed with the study.

Q1: Are You (Already) Vibing?

The session began with a question to the audience: *"Are you (already) vibing?"* Even before explaining what vibe coding meant, **43 attendees said yes, and 18 said no (**Figure 1**)**. The term 'vibe coding', coined by Andrej Karpathy, refers to a new mode of development where engineers collaborate with LLMs in a semi-structured, conversational, and flow-based way [1].

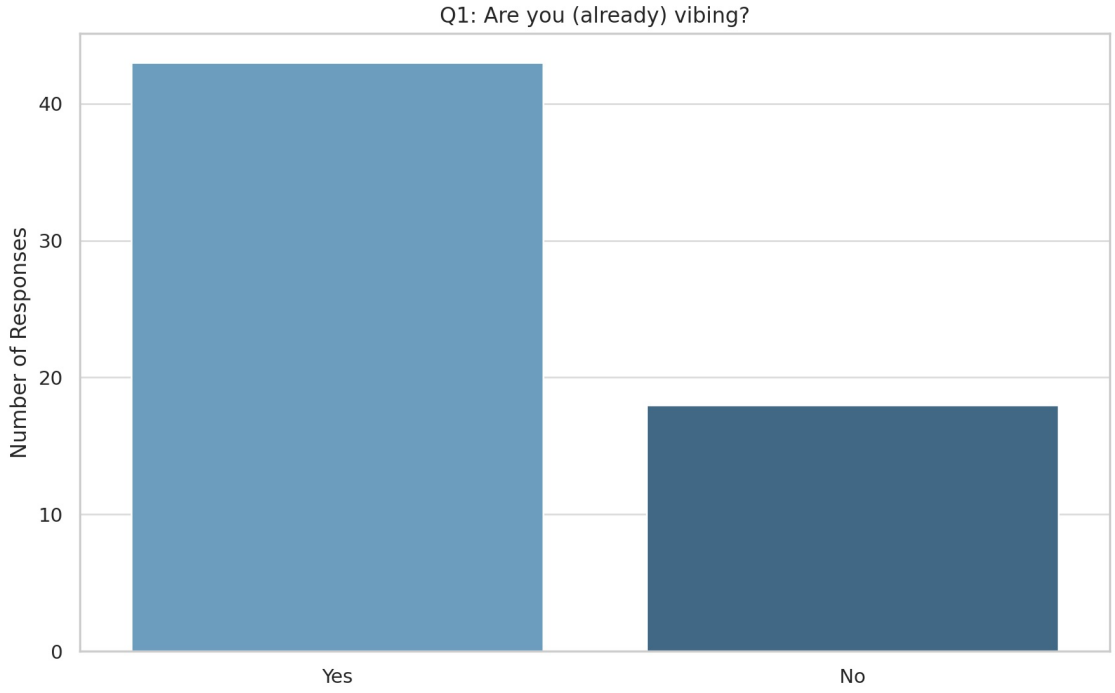

Figure 1 – Results for Question 1 - "Are you (already) vibing?"

This opening question was deliberately provocative. Beyond engaging the audience, it established a narrative bridge toward the central theme of the session: reconciling the intuitive, emergent practice of *vibe coding* with the structured, traceable rigor of the **Software Development Life Cycle (SDLC)**. This transition emphasized the ongoing importance of SDLC principles, even as LLMs introduce new levels of fluidity and abstraction into the development process.

Q2: What LLMs Are You Using? For What Software Engineering Task? And In What Context (Pet Project or Work)?

Through a set of fictional team personas: **Anna** (“The Codebase Queen”: tech lead and architect), **Axel** (“Bugsy Rookie”: junior tester with attention to detail), and **Ari** (“Captain Backlog”: product owner enabling the team), the talk showed how different roles are engaging with tools like ChatGPT, Claude, GitHub Copilot, and more. For instance:

- Anna sometimes pulls in Claude for deeper architectural brainstorming or to give constructive, encouraging feedback on a pull request from a junior developer.
- Axel refers to GitHub Copilot as his go-to for test design and achieving code coverage. It’s great at helping him think in steps and logic, not just generating code, but guiding him through the why.
- Ari is the sharp, people-savvy “Captain Backlog”, that uses ChapGTP to master backlog grooming and user story writing**.**

These examples raised a key question, where the audience was asked to name their LLM of choice, the type of SE task they were using it for, and whether it was used for work or a pet project. **189 responses from 123 participants were collected**.

The responses are summarized in Figure 2. Participants reported using a variety of tools, with the most common being: **ChatGPT, Claude and GitHub Copilot**. Regarding the most recurring Software Engineering (SE) Tasks: **Code generation** emerged as the most frequently mentioned task**.** Respondents were evenly split, but a slight majority indicated **work-related use**, while **pet projects** were also frequently mentioned, highlighting the experimentation space LLMs currently occupy.

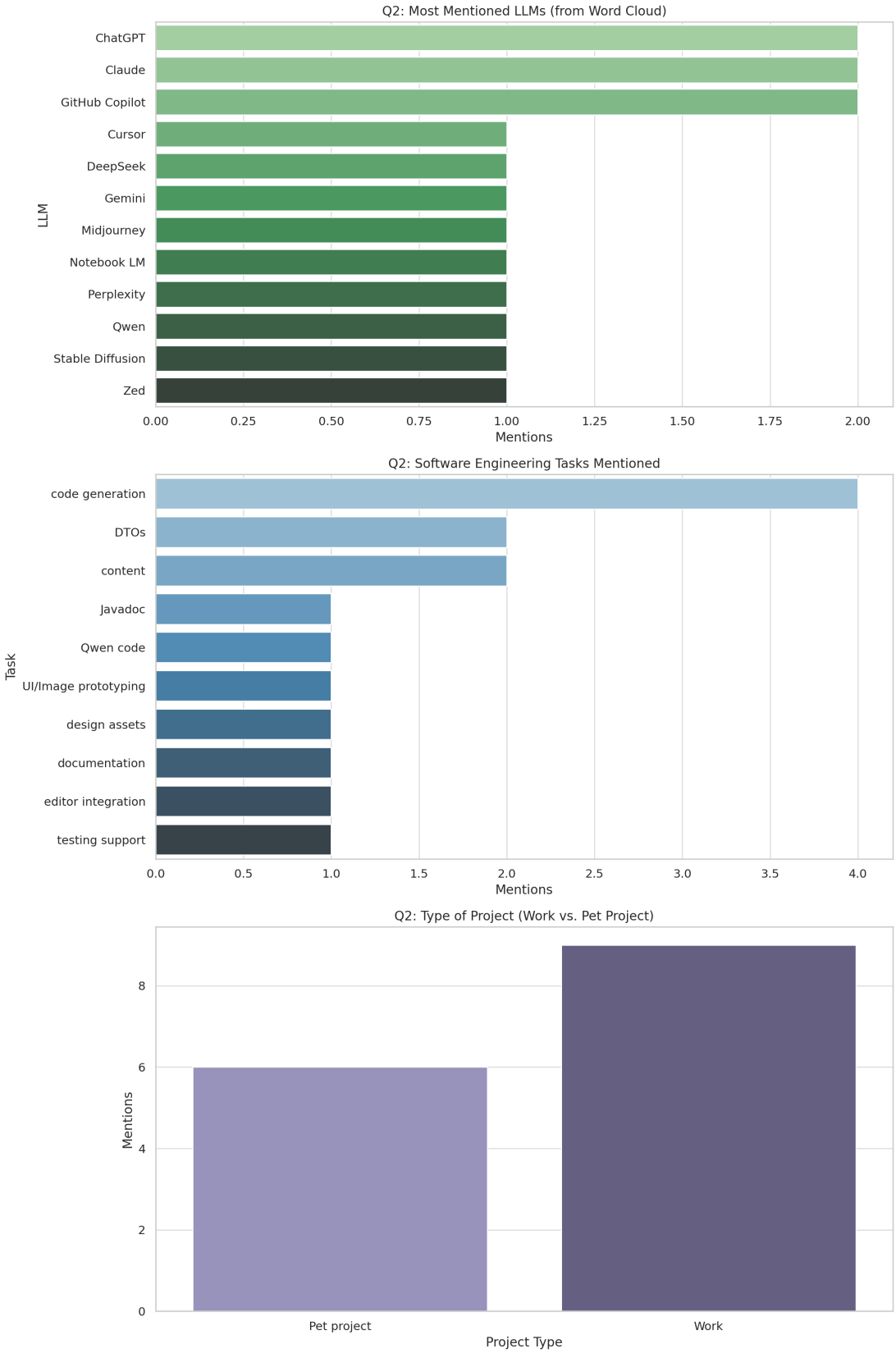

Figure 2 - Results for Question 2 – LLM of Choice + SE task + PetOrWork Project

Q3: Where are you on the vibe curve?

To deepen the conversation, the "Vibe Coding Hype Cycle", powered by ChatGPT - a playful but insightful adaptation of the Gartner® Hype Cycle, was presented. This model helped frame how individuals and teams experience LLM adoption, across five key and sequential stages (Figure 3):

1. Innovation Trigger – just discovered LLMs, curious but unsure.
2. Peak of Inflated Expectations – everything seems magically solvable.
3. Trough of Disillusionment – realizing limitations, hallucinations, and frictions.
4. Slope of Enlightenment – learning how to prompt, evaluate, and adapt.
5. Plateau of Productivity – sustainable, integrated use in everyday development.

The usual timeframe for hype cycles to reach their last stage is usually between 3 to 5 years.

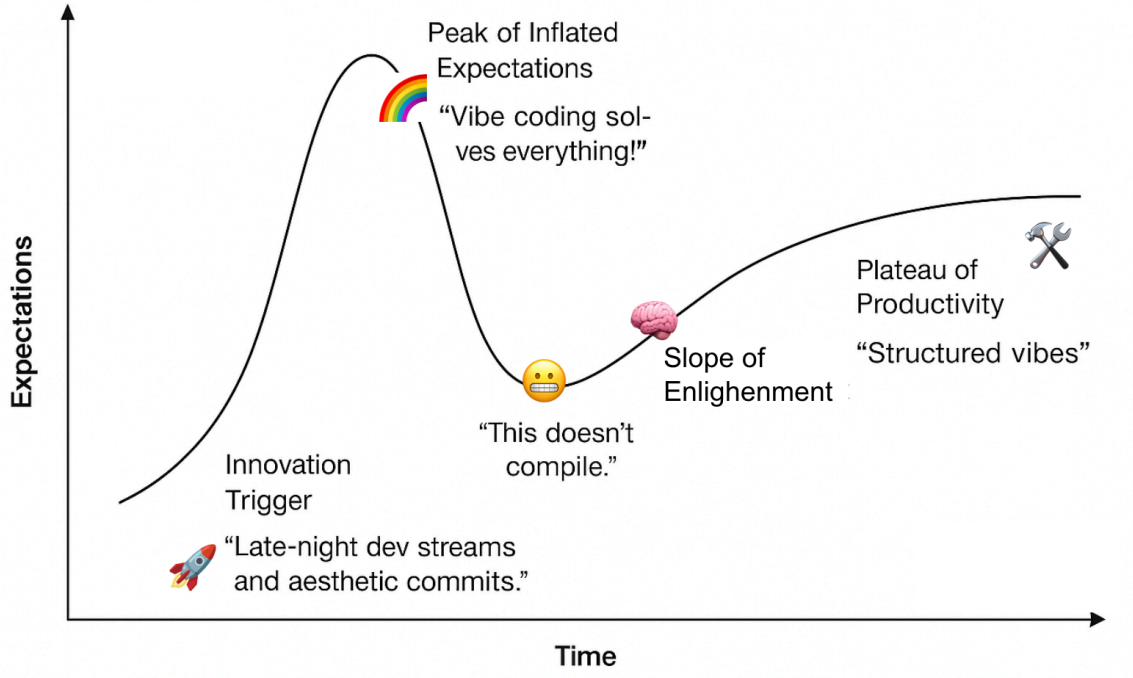

Figure 3 – "Vibe Coding" Hype Cycle (powered by ChatGPT)

When asked: *"Where are you on the vibe curve?"* **108 participants** responded, with most placing themselves in the Slope of Enlightenment (Figure 4).

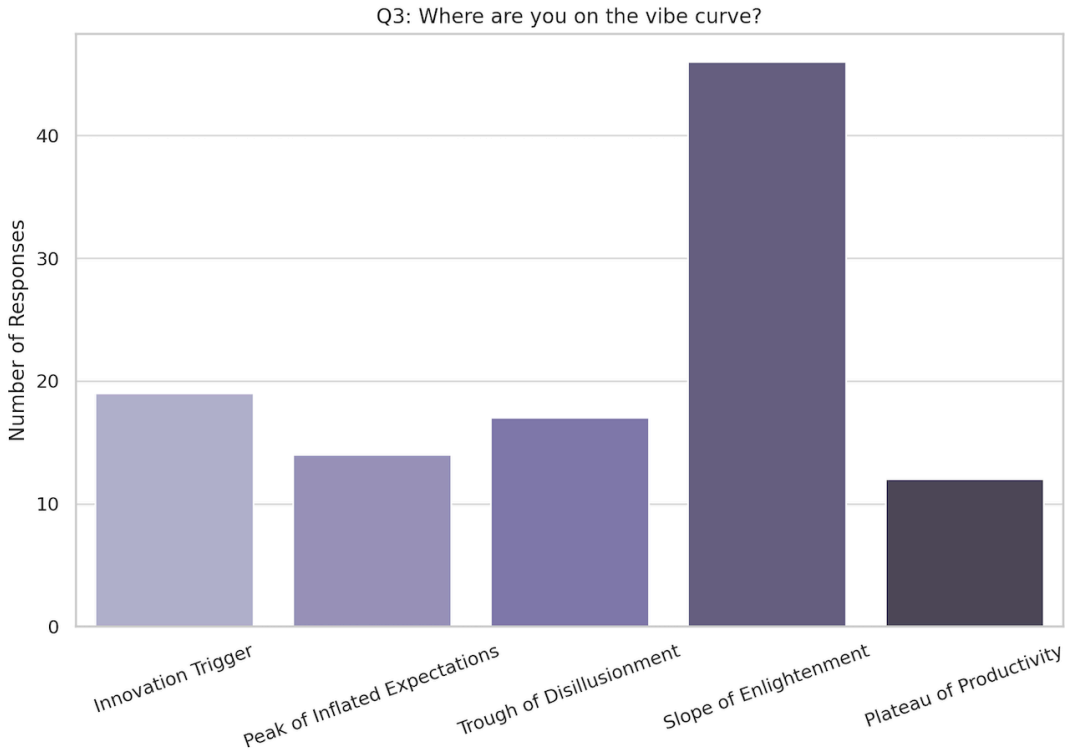

Figure 4 – Results for Question 3 – "Where are you on the vibe curve?"

Research-Based Framing

The session presented three key reference papers to explore what the scientific literature reveals about the impact of LLMs on the software development lifecycle and productivity.

First, a systematic review of 395 studies from 2017–2024, identified the usage of 70 distinct LLMs used for over 85 software engineering tasks [3]. These include code summarization, code completion, program synthesis, and vulnerability detection. The findings indicate that the LLMs play crucial roles in automating and optimizing these tasks, thereby enhancing developer efficiency and accuracy. Furthermore, the research addresses challenges associated with effectively deploying LLMs in software engineering contexts. It outlines the complexities related to training these models, which require significant computational resources and custom datasets, such as GitHub repositories, bug reports, API documentation, and Q&A pairs from resources like Stack Overflow. The paper emphasizes that while the potential for LLMs in SE is considerable,

careful consideration of their implementation and evaluation is essential for facilitating future advancements.

"Lost in the Middle: How Language Models Use Long Contexts" by Liu investigates the capacity of language models to effectively process lengthy contexts [4]. Findings reveal that these models demonstrate a recency bias, significantly favoring information from the beginning and end of the context while being less adept at using relevant content located in the middle. This performance pattern is evidenced by a U-shaped curve indicating the models decreased accuracy when tasked with extracting information that is situated away from these boundary positions. This raises questions regarding the implications of model architecture on context utilization, indicating that advancements in long-context models may require strategies like document reranking or input context reconfiguration to mitigate limitations presented by the current biases. The study underlines the necessity for improved methodologies in handling long input contexts to enhance the performance of modern language models on tasks requiring extensive contextual understanding.

Lastly, "The Effects of Generative AI on High-Skilled Work: Evidence from Three Field Experiments with Software Developers" analyzes the influence of an AI-powered coding assistant on the productivity of software developers through randomized controlled experiments conducted at Microsoft, Accenture, and a Fortune 100 company [5]. The results indicate a noteworthy productivity increase among developers using the AI tool. While the figure of 26.08% (SE: 10.3%) may reflect a specific analysis, it must be noted that the detailed statistics were not provided in the available reference. Productivity was specifically measured by comparing the number of completed tasks (e.g. Number of commits, pull requests…) between developers who had access to the AI tool and those who did not. The work highlighted that factors influencing technology adoption such as user satisfaction and perceived quality, further affect productivity outcomes.

In conclusion, LLMs provide a significant leap in automating and enhancing key components of the SDLC, promising increased productivity and better-quality software outputs, while also presenting avenues for further investigation to overcome existing challenges. However, and despite these benefits, challenges persist in leveraging LLMs effectively within the SDLC. Issues around data privacy, the need for tailored training

datasets, issues with the architecture and context, and metrics for evaluating LLM performance and productivity of the teams and individuals are areas that require further research and development. The complexity of assessing qualitative improvements brought by LLMs needs attention, as current metrics may fall short of capturing their full potential.

Conclusions

The session highlighted several pressing challenges surrounding the integration of LLMs into the software development lifecycle. A key concern is how to define and measure productivity, with current metrics often failing to capture the nuanced impact of these tools. Model size and deployment introduce additional complexity, particularly when balancing performance with cost. Questions also remain around the use of general-purpose versus domain-specific models, each presenting unique trade-offs. Reliable assessment methods are still evolving, making it difficult to evaluate outcomes consistently. Concerns related to explainability, interpretability, trust, and ethics are increasingly urgent, especially as LLMs influence critical decision-making processes. The risks to software security and the growing trend of open washing, where tools are marketed as open but lack transparency, further complicate adoption. Lastly, the lack of structured training opportunities in academia poses a barrier to building the skills and mindsets needed to responsibly leverage these technologies.

Before You Go

- Train your prompt engineering skills (“vibe along” 😊)
- Ask for the reasoning of the responses (eg. Chain-of-Thought)
- Be (stay) curious, skeptical, and experimental
- Test, test, test

References

[1] C. I. Reis, "Exploring the Potential of LLMs in the SDLC: A Productivity Perspective – slidedeck", JNation2025, 27 and 28 May (2025)", accessed: 06-Jun-2025. [Online]. Available: osf.io/c3vbz.

[2] Karpathy A. - X, (2025), "There's a new kind of coding I call "vibe coding", https://x.com/karpathy/status/1886192184808149383 (accessed May 20, 2025).

[3] X. Hou, Y. Zhao, Y. Liu, Z. Yang, K. Wang, L. Liet al., "Large Language Models for Software Engineering: a Systematic Literature Review", ACM Transactions on Software Engineering and Methodology, vol. 33, no. 8, p. 1-79, 2024. https://doi.org/10.1145/3695988

[4] N. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroniet al., "Lost in the Middle: How Language Models Use Long Contexts", Transactions of the Association for Computational Linguistics, vol. 12, p. 157-173, 2024. https://doi.org/10.1162/tacl_a_00638

[5] Z. Cui, M. Demirer, S. Jaffe, L. Musolff, S. Peng, & T. Salz, "The Effects of Generative AI on High-skilled Work: Evidence From Three Field Experiments with Software Developers", 2024. https://doi.org/10.2139/ssrn.4945566