Towards Capturing Software Purpose in AI-Native Software Development

Daniel Planötscher[1,*]

[1]Free University of Bolzano, via Bruno Buozzi 1, Bolzano, 39100, Italy

Abstract

As generative AI (GenAI) is increasingly adopted in software development, an AI-native approach to software development is emerging. In this approach, we are not only integrating AI into current software development practices and workflows but also entirely rethinking our software development life cycle to collaborate with GenAI more effectively and enhance human capabilities through GenAI. In this context, new paradigms of software development, such as vibe coding, are emerging as intent-driven approaches to designing and implementing software. Here, the developer provides the intent of the system's expected functionalities and behaviours, and the AI generates the application code independently. However, these approaches often neglect the potential for AI collaboration in the earlier, critical phase of exploring and defining the needs and problems that the application must address. This problem-definition stage is essential to problem-solving, as it establishes the system's purpose and objectives. AI could offer significant value here by synthesising information, enhancing divergent brainstorming, and more. As a result, this paper defines research questions exploring how AI might be integrated into the early problem-definition phase of software development and proposes the design thinking philosophy as a possible direction for this integration.

Keywords

AI-Native Software Development, Human-AI Collaboration, Vibe Coding, Design Thinking

1. Background

In the last few years, the field of software development has undergone several changes. Ever since generative AI (GenAI) tooling was first introduced in 2022, with the public announcement of both GitHub Copilot [1] and ChatGPT [2], they have experienced widespread adoption among practitioners [3]. In the beginning, GenAI-based tooling was mainly valued for increasing productivity, for instance, by assisting with everyday tasks such as refactoring, requirements management, getting explanations, and more [1]. GenAI was soon used to generate entire functions in software code or to create intelligent, context-aware auto-completions [2, 3, 1], significantly speeding up software development. As a result, AI was soon seen as a valuable companion with a second pair of hands that could take over some tasks and assist with daily work, enhancing the way of working. The main driver, however, was still the human, delegating the AI smaller tasks so they could focus on more important things.

Nevertheless, times are changing. With the recent successes of AI agents and their ability to execute high-level tasks independently, new ways to leverage their capabilities in software development are being explored. In this context, we can see agentic development environments emerging, such as Cursor[4], that demonstrate these capabilities first-hand [4, 5], by independently visualising the current code repository and implementing and executing code changes. These tools demonstrate the potential of AI working alongside humans without requiring detailed guidance, and they can develop sophisticated functionality with a simple natural language prompt.

ICSOB '25: 16th International Conference on Software Business, November 24–26, 2025, Stuttgart, Germany

*Corresponding author.

dplanoetscher@unibz.it (D. Planötscher)

0009-0001-9162-0939 (D. Planötscher)

[1]https://github.blog/news-insights/product-news/github-copilot-is-generally-available-to-all-developers/

[2]https://openai.com/index/chatgpt/

[3]https://survey.stackoverflow.co/2023/

[4]https://cursor.com/

With these new advances in AI-driven software development, a new era is dawning on the horizon. As AI agents are increasingly leveraged by developers[5], doubts are emerging about whether using AI only to enhance existing processes is the best way to leverage it [3]. This has sparked a discussion, hinting at a more AI-native approach to software development that aims to rethink the process from the ground up [3]. Instead of simply introducing GenAI into existing processes and enhancing small tasks wherever possible, workflows and practices should be adapted to leverage AI's benefits more effectively. In this case, GenAI is not only seen as an add-on to the existing process but also as a starting point for a new way of building software, defining new roles and approaches, entirely changing how software development could and should be performed. With this, GenAI should be seamlessly integrated into the whole process and seen as a crucial part of every phase of the development life cycle [6].

In this emerging landscape of AI-native software development, new paradigms for building software are coming to light, such as *vibe coding* and *spec-driven development*[6]. Vibe coding is a term first coined by Andrej Karpathy in February 2025 [7]. In this approach, software developers use generative AI tooling to create entire applications by describing the intended functionality in natural-language prompts, entrusting the whole implementation to AI agents. The generated source code is thereby completely disregarded and managed only by the AI, allowing the human developer to focus solely on the outgoing application [8]. This enables the creation of working prototypes in under 10 minutes by simply providing the AI with a general idea of an application and iteratively refining the results [8]. Spec-driven development, on the other hand, focuses on using AI to implement well-defined specs developed in collaboration with it, creating more predictable, reliable, and maintainable software that goes beyond simple prototypes to near-production-ready code.

Nevertheless, across these approaches to AI-native software development, we can observe that they are typically described as intent-first [3]. For instance, in vibe coding, software development is seen as a co-creative flow, where the human expresses the intent of what the software should do in natural language, and the AI generates the software artefact and thereby the final implementation code [9]. In spec-driven development, the human has to provide the expected user journeys, functionalities and behaviours to the AI system, which is then refined into specifications and later implemented working software code. As a result, in both cases, the developer already has to give the presumed solution path, and the AI implements it, mixing in its own ideas and assumptions when details are missing, allowing the developer and the AI to iterate on a solution together.

2. Problem

This, however, is also where the limitation of current approaches to AI-native software development lies. In the real world, software development typically starts with exploring and better understanding the problem or the need the application aims to address. With this, software developers can develop solutions that actually align with users' needs and therefore solve their problems. Currently, AI-native approaches to software development incorporate AI only into the design and implementation of the software application's final solution, letting the developer focus on identifying possible solution paths and using AI only as guidance to iterate on the solution. The developer typically provides solution paths as the intent they have in mind by writing a prompt that defines the application's idea or the user journeys they have in mind. The underlying intent is guided by the software application's purpose and formalised into lower-level goals that the AI needs to implement to address the needs the application aims to address.

Consider a concrete example: A developer prompts an AI with "Create a task management app with priorities, due dates, and calendar integration." The AI immediately implements this solution without questioning whether users actually require another task manager. Suppose the underlying problem is task overload and decision paralysis, rather than organisational issues. In that case, the AI may develop

[5] https://survey.stackoverflow.co/2025/ai

[6] https://github.blog/ai-and-ml/generative-ai/spec-driven-development-with-ai-get-started-with-a-new-open-source-toolkit/

an application that functions as specified but fails to address the actual user need.

In fact, understanding and defining a need and a problem is often seen as one of the most challenging yet crucial parts of software development. Not understanding the problem you are trying to solve, or creating solutions that do not address the core issues of users and customers, are common failure factors in software projects. For instance, in the startup world, it is very common to encounter problems in the problem-solution fit of software products [10], such as by addressing issues that do not exist in the real world, or developing solutions that do not solve the problems they aim to tackle in the first place. Yet, in traditional software projects, the role of requirements and the challenges posed by misunderstood or poorly captured requirements are well known [11], and these are also common reasons for project failure [12].

Even though this is a common problem in software development, AI-native software development nowadays places the least focus on it. While a lot of tool support and assistance exists for the design and development of the final solution, little emphasis is placed on defining and exploring the needs and problems, and on the resulting application purpose, which determines what the application is and what it looks like at the end.

There are several reasons for this disregard of this part of the software development process. This part of the software development process is often perceived as the more "human" part, i.e., where empathy, domain knowledge, and strong customer collaboration might be necessary factors that AI might not fully possess. It is the part where experience is essential, and a lot is driven by contextual knowledge and understanding of the customers and the world they live in.

Still, this underestimates the potential of GenAI to assist in this part of the software development process. Even if GenAI might lack genuine empathy and lived experience, it could still help by contributing to creating software that is better aligned with customers. GenAI might help synthesise information from several sources and formalise concrete insights into problems and needs. It might help in divergence and brainstorming, providing novel ideas based on the provided issues. Then it might help in designing the solutions based on the information provided and implementing prototypes, again synthesising feedback and deriving learning to align the solution even better. Everything in strict collaboration with humans, symbiotically enhancing each other's strengths and weaknesses.

As a result, when developing an AI-native software development process, it might be essential to explore how AI can be introduced throughout the problem-solving process, from defining the problem to brainstorming and exploring different solution paths, to developing and refining solutions.

3. Research Objectives and Proposed Approach

The gap identified above raises fundamental questions about how AI-native software development should evolve. Considering that current approaches primarily focus on solution design and implementation while neglecting problem exploration and definition, this raises several questions about the future of AI-native software development.

As a result, this paper proposes exploring the following research questions:

- **RQ1:** Is it feasible and meaningful to introduce GenAI into the problem exploration and definition phases of software development, or are these phases inherently too dependent on human judgement?
- **RQ2:** What could AI-native software development look like when introducing GenAI into the purpose exploration and definition of the system? How does it change the roles and practices of software development approaches?
- **RQ3:** What role can GenAI play in the exploration and definition of the purpose of a software system?
- **RQ4:** What patterns of human-AI collaboration emerge when GenAI is involved in problem definition versus solution implementation?
- **RQ5:** How does incorporating GenAI into problem definition stages impact the quality and user-alignment of resulting software solutions?

To address the gap identified above, this paper proposes Design Thinking as a theoretical framework for reimagining AI-native software development. Design thinking is defined as "a non-linear, iterative process that teams use to understand users, challenge assumptions, redefine problems and create innovative solutions to prototype and test"[7]. It fits perfectly into the gap defined in this paper, as it establishes a philosophy for understanding and determining problems, and for solving them without sacrificing user-centricity and innovation. As a result, it could serve as a guiding mindset for defining an AI-native software development process that leverages AI throughout the entire problem-solving process.

Five stages define Design Thinking [8]:

1. **Empathise**—first, the problem is better understood, commonly through user research and setting aside existing assumptions.
2. **Define**—After understanding the problem, it is formalised and better defined, through synthesising the observations and defining problem statements and user personas.
3. **Ideate**—Only after having a clearer idea of the problem, solution paths are brainstormed and designed, especially by thinking outside the box and coming up with innovative solutions.
4. **Prototype**—In the end, the defined solution path is implemented into a prototype, to experiment with the approach and get feedback that fosters learning.
5. **Test**—The prototype is given to real users, who can provide real-world feedback and learnings.

One possible research direction is to examine how GenAI can be integrated into each stage of this process, but not only as a tool within existing approaches, but to redefine how each of these phases is practised. This may involve adapting traditional techniques such as personas or journey mapping for AI collaboration, or developing entirely new methods better suited to how AI creates, processes and synthesises information.

4. Conclusion

This paper reflects on current approaches to AI-native software development and shows that the most crucial and challenging aspect of software development is often overlooked. As such, it proposes introducing the exploration and definition of a software system's purpose into AI-native software development processes to build software that is better aligned with users. The paper investigated how AI might help in these phases by synthesising and formalising users' problems more effectively, and by assisting with divergent brainstorming and defining possible solution paths. With this, AI might not only be introduced into this phase of the software development process but also enhance it, which is a crucial success factor for practical software projects.

To explore this further, this paper proposes several research questions that investigate how AI might be implemented in this phase of the software development life cycle and how it might affect execution and quality. It also proposes a possible direction, introducing Design Thinking as a philosophy for developing better-aligned solutions and for exploring problems more effectively and human-centricly.

Ultimately, we are still in the early days of AI-native software development and are only beginning to understand what this future might look like. As such, it may be necessary to fundamentally reevaluate what the central aspects of software development are and what truly defines the viability and success of a software application. Building software that truly solves real problems and addresses genuine user needs might be the most essential aspect of software development, yet current AI-native approaches largely overlook this phase. As a result, the future of AI-native software development should focus not merely on how fast we can build, but on ensuring we make the right things. Human-AI collaboration might enable us to create solutions that are genuinely aligned with user needs, working together from problem exploration through to implementation.

[7] https://www.interaction-design.org/literature/topics/design-thinking
[8] https://www.interaction-design.org/literature/article/5-stages-in-the-design-thinking-process

Declaration on Generative AI

During the preparation of this work, the authors used Microsoft Copilot in order to: Grammar and spelling check. After using this tool, the authors reviewed and edited the content as needed and take(s) full responsibility for the publication's content.

References

[1] K. Stutz, K. Sandkuhl, M. Möhring, Empirical insights into the usage of generative ai in software engineering, in: International Conference on Business Informatics Research, Springer, 2025, pp. 257–271.

[2] H. Li, H. Zhang, A. E. Hassan, The rise of ai teammates in software engineering (se) 3.0: How autonomous coding agents are reshaping software engineering, arXiv preprint arXiv:2507.15003 (2025).

[3] A. E. Hassan, G. A. Oliva, D. Lin, B. Chen, Z. Ming, et al., Towards ai-native software engineering (se 3.0): A vision and a challenge roadmap, arXiv preprint arXiv:2410.06107 (2024).

[4] D. B. Acharya, K. Kuppan, B. Divya, Agentic ai: Autonomous intelligence for complex goals–a comprehensive survey, IEEe Access (2025).

[5] R. Sapkota, K. I. Roumeliotis, M. Karkee, Ai agents vs. agentic ai: A conceptual taxonomy, applications and challenges, arXiv preprint arXiv:2505.10468 (2025).

[6] C. Hymel, The ai-native software development lifecycle: A theoretical and practical new methodology, arXiv preprint arXiv:2408.03416 (2024).

[7] A. Sarkar, I. Drosos, Vibe coding: programming through conversation with artificial intelligence, arXiv preprint arXiv:2506.23253 (2025).

[8] A. Osmani, Beyond Vibe Coding: From Coder to AI-Era Developer, O'Reilly Media, 2025. URL: https://www.oreilly.com/library/view/beyond-vibe-coding/9798341634749/.

[9] C. Meske, T. Hermanns, E. von der Weiden, K.-U. Loser, T. Berger, Vibe coding as a reconfiguration of intent mediation in software development: Definition, implications, and research agenda, arXiv preprint arXiv:2507.21928 (2025).

[10] C. Giardino, X. Wang, P. Abrahamsson, Why early-stage software startups fail: a behavioral framework, in: International conference of software business, Springer, 2014, pp. 27–41.

[11] A. Hussain, E. O. Mkpojiogu, F. M. Kamal, The role of requirements in the success or failure of software projects, International Review of Management and Marketing 6 (2016) 306–311.

[12] J. Verner, J. Sampson, N. Cerpa, What factors lead to software project failure?, in: 2008 second international conference on research challenges in information science, IEEE, 2008, pp. 71–80.