Personalizing LLM-Based Conversational Programming Assistants

Jonan Richards
jonan.richards@ru.nl
Radboud University
Nijmegen, The Netherlands

Abstract

Large Language Models (LLMs) have shown much promise in powering a variety of software engineering (SE) tools. Offering natural language as an intuitive interaction mechanism, LLMs have recently been employed as conversational "programming assistants" capable of supporting several SE activities simultaneously. As with any SE tool, it is crucial that these assistants effectively meet developers' needs. Recent studies have shown addressing this challenge is complicated by the variety in developers' needs, and the ambiguous and unbounded nature of conversational interaction. This paper discusses our current and future work towards characterizing how diversity in cognition and organizational context impacts developers' needs, and exploring personalization as a means of improving the inclusivity of LLM-based conversational programming assistants.

Keywords

SE, LLM, HCI, conversational agent, personalization

ACM Reference Format:
Jonan Richards. 20XX. Personalizing LLM-Based Conversational Programming Assistants. In *Proceedings of Doctoral and Early Career Symposium of the 19th International Conference on Cooperative and Human Aspects of Software Engineering (CHASE 2026 DECS).* ACM, New York, NY, USA, 5 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Large Language Models (LLMs) have shown much promise in performing a variety of code-centric software engineering (SE) tasks such as code generation, bug detection, and in supporting developers' code understanding [9]. LLM-based tools that are implemented as conversational assistants, hereafter referred to as "programming assistants", hold potential for SE by not only providing technical utility but also supporting human aspects of software development. Programming assistants can support several SE activities at once [20], can increase productivity gains by allowing iterative refinement and progress towards the developer's goal [1], and is missed by developers when not present [13]. Examples of programming assistants include in-IDE chat functionalities of GitHub Copilot[1] and Tabnine[2], but also encompass more general assistants such as ChatGPT[3] which are often used by developers in coding contexts [23].

Despite their apparent potential, several links have been found between challenges in using programming assistants and developers' experience, learning style, and gender [5, 14, 15, 21], which may partially explain the gender and experience gap in the adoption of LLM-based tools [6]. While it is well-known that cognitive diversity, which has been linked to gender, affects people's software-based problem-solving styles [3], this evidence suggests that it also affect developers' needs and behavior when interacting with programming assistants. Besides cognition, organizational factors such as policies, team dynamics, and the position of developers along the software development lifecycle also impact their needs and challenges they run into [10].

LLMs can inherently tailor outputs to personal preferences by allowing for natural language interaction, which developers greatly appreciate [13, 21]. However, effectively prompting programming assistants requires substantial effort and experience [13–15]. Clearly, this inherent personalized support has a high barrier to entry, and the challenges faced by only specific groups of developers still highlight a lack of support for cognitive diversity, including experience and problem-solving styles. This line of research is aimed at characterizing the impact of cognitive diversity and organizational factors on individual developers' needs and interaction style regarding programming assistants, and designing new personalization approaches to increase inclusivity of these tools.

> **RQ** – How can an LLM-based programming assistant provide effective personalized support to increase inclusivity to cognitive diversity and organizational context?

The main contributions of this research will be 1) an explanatory framework of how cognitive and organizational factors affect developers' needs and interaction styles, 2) the design of personalization strategies to improve inclusivity towards these factors, and 3) a personalized programming assistant prototype. Together, these contributions will provide researchers with insights on developer behavior, support developers through tailored interactions, inform programming assistant providers in designing more inclusive support, and help organizations deploy tools that better accommodate diverse developers.

CHASE 2026 DECS, Rio de Janeiro, Brazil

ACM ISBN 978-1-4503-XXXX-X/20XX/XX
https://doi.org/XXXXXXX.XXXXXXX

[1] https://github.com/features/copilot
[2] https://www.tabnine.com/ai-chat/
[3] https://openai.com/index/chatgpt/

2 Background

This research builds on the existing body of human-computer interaction (HCI) research. User experience (UX) is often decomposed into pragmatic UX (e.g. usefulness and usability) and hedonic UX (e.g. enjoyment, autonomy) [17]. According to Hassenzahl's model of UX [8], hedonic qualities arise from how well a product enables the user's basic psychological needs and the resulting "be-goals", i.e. how the user wants to be or feel (such as in control, knowledgeable, supported). These "be-goals" give rise to users' "do-goals" (what they want to do using the product), and the extent to which these are enabled by the product results in the pragmatic experience. Norman's Seven Stages of Action model [16] describes the cyclical process of how these "do-goals" are translated into intentions and in turn into actions which are then executed. Next, the interactive system's results are perceived, interpreted and evaluated with respect to the goals and intentions, possibly leading to revised intentions and actions. Subramonyam et al. [22] explore challenges users might face in this process when interacting with LLMs, including the setting of goals and intentions so that LLMs can accomplish the task, translating their intentions into optimal actions (prompts), and knowing what to expect from LLMs' output.

It is commonly accepted in HCI research that both personal factors (e.g. cognitive factors, experience with a product, attitude) and contextual factors (e.g. social and organizational context) greatly impact users' needs, behavior, and the challenges they face [2, 3]. Personalization is a commonly used strategy to ensure information systems cater to the wide range of user needs and behavior arising from diversity in personal and contextual factors [7]. However, personalization encompasses a variety of approaches in practice: whereas adaptive systems employ implicit personalization to user behavior and user needs inferred from this behavior, adaptable systems employ explicit personalization by allowing users to configure the system to their own needs [7].

3 Research Overview

In this research, I refer to developers' psychological needs and their "be-goals" as their *needs*, and to the cyclical process of transforming goals into intentions and actions (i.e. behavior) as their *interaction style*. The resulting relationships between these concepts and personal and contextual factors are shown in figure 1. As a hypothetical example, take a junior developer, new to a company, using a programming assistant. The personal and contextual factors include them having relatively little programming experience, and using the assistant in an on-boarding context. These factors could affect their needs, prioritizing their need for becoming familiar the code base over the need for productivity. This need for learning may induce the developer to adopt an interaction style where they ask the assistant to explain the code base and give pointers to a solution to the task at hand, rather than just provide the solution right away. The developer's personal factors may influence their interaction style as well, as their low programming experience could lead them to phrase their questions in a more abstract and high-level way than an experienced developer might.

To improve both pragmatic and hedonic UX, a personalized programming assistant will need to take into account developers' needs, and accommodate developers' challenges in interacting by

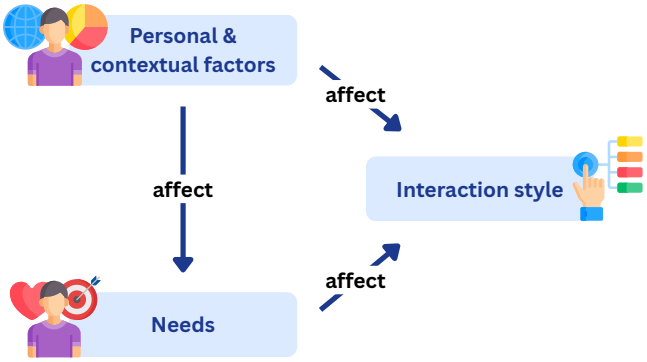

Figure 1: Conceptual model of diversity in interaction used in this research.

considering their interaction style. A developer's interaction style is directly observable to a programming assistant, but their needs will need to be inferred from their interaction style or specified through human-in-the loop methods. We will explore both these implicit and explicit approaches. Figure 2 shows an overview of the planned phases for this research, from understanding diversity in interactions (phase I), to designing personalization approaches (phase II) and implementing and evaluating a prototype assistant (phase III).

Since the research involves human subjects, ethical approval will be obtained from the relevant ethics board before each phase.

3.1 Preliminary Study

> **RQ_{PRE}** – Is automatic personalization of programming assistants feasible?

To assess the feasibility of fully automatic personalization and answer **RQ_{PRE}**, we ran a preliminary study, **Study PRE** [18]. We decided to initially focus only on personal factors by fixing the context of the experiment to a code understanding setting. In this study, we designed a code understanding assistant prototype that infers developers' needs, leveraging established prompting techniques to improve LLMs' Theory of Mind (the ability to reason about others' mental states). Due to the exploratory nature of the study, no explicit guidance was given to the LLM on the relevant aspects of developers' mental state, or on how to use the inferred needs to adapt the support accordingly.

We evaluated the approach by comparing it to a non-personalized assistant prototype in a within-subject study with fourteen novice developers to capture their perceptions and preferences. Personalization was perceived to improve the assistant's understanding of novices' queries and the clarity of its responses. We also found that participants' interaction styles with the assistants varied significantly, showing that even the constrained participant sample of novices exhibited great diversity. Furthermore, different interaction styles were shown to significantly impact the personalized assistant's effectiveness. Although we demonstrated the feasibility of personalizing a code understanding assistant, it is clear LLMs need explicit guidance on how to personalize. To provide this guidance, we first need a thorough understanding of the interplay between

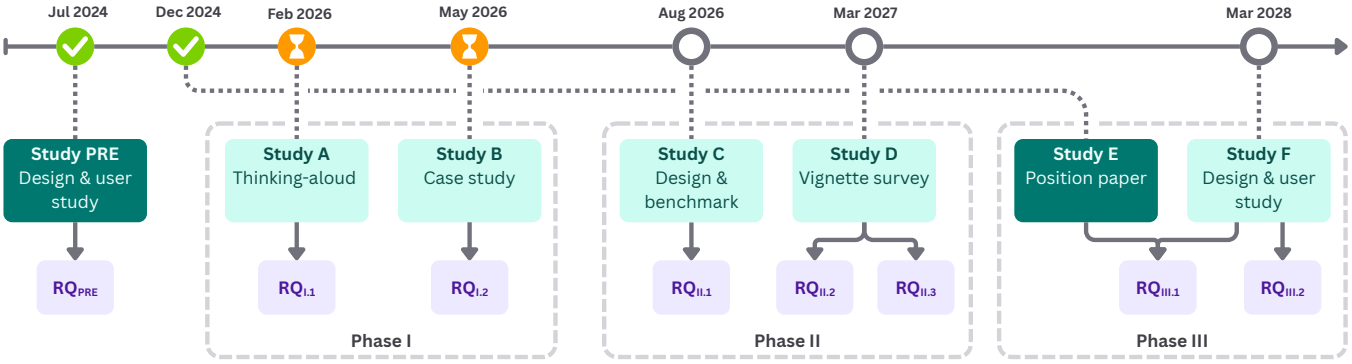

Figure 2: Overview of research phases, including a timeline for completion.

developers' personal and contextual factors, needs, and interaction style.

3.2 Phase I – Understanding Diversity in Programming Assistant Interactions

> **RQI.1** – How does experience and cognitive style affect developers' needs and interaction style in programming assistant interactions?

> **RQI.2** – How does organizational context affect developers' needs and interaction style?

In the initial phase of this research, we attempt to understand the effect diversity has on developers' interactions with programming assistants. Although intersectionality likely plays a role, we decided to study the impact of personal and contextual factors separately. This keeps methodological complexity within feasible limits, while allowing us to gain insights into the extent and patterns of diversity in interactions, as well as better ways to support it.

Although there is preliminary evidence of usability issues in programming assistants faced by certain groups of developers [5, 14, 15], a thorough understanding of such issues and their underlying causes is needed before we can detect or address them. In **Study A**, which is currently being conducted, we aim to characterize the relationships between diversity in developers' personal factors, needs, and interaction style. To this end, we have conducted a user study with $n = 27$ participants with diverse experience and cognitive profiles. We disregard contextual factors by fixing the experimental context to a code change setting. In a series of progressively harder code change tasks in a single codebase, supported by the programming assistant *Github Copilot Chat*, we combined a thinking aloud setup with post-study semi-structured interviews and questionnaires to get insights into participants' cognitive processes, needs, and challenges they faced. A quantitative analysis of chat messages to Copilot was conducted to identify interaction styles. Currently, these are being related to participants' cognitive and experience profiles to answer **RQI.1**. Preliminary results include distinct needs, interaction styles, and preferences regarding Copilot's support, regarding e.g. the presence of an explanation when generating code, the structure and depth of guidance, and the use of technical terms. These individual differences reinforce the need for tailored support.

There are several existing studies on the effect of contextual factors such as social factors (e.g. support from colleagues, organizations, online communities) [4, 21], organizational policies [10] and cultural values [12] on developers' trust and adoption of programming assistants. However, these studies are relatively high-level, not focusing on how contextual factors shape interaction styles beyond whether or not developers use programming assistants. Moreover, participants are recruited across companies meaning that dynamics within companies and teams may go unnoticed, such as the impact of one developer's tool use on others and whether views on the use of these tools differ between individuals in the same context. In a case study at a software engineering company for **Study B**, we will address this gap by conducting semi-structured interviews. By mapping how social and organizational context affects needs, attitudes, and interaction styles with programming assistants vary and interconnect between roles and individuals in an organizational context, we will not only answer **RQI.2** but also gain insights for practitioners and managers on how to increase productivity and decrease friction within software engineering teams.

3.3 Phase II – Designing Personalization Approaches to Address Diversity

> **RQII.1** – How can developers' needs be inferred from their interaction style with a programming assistant?

> **RQII.2** – How do varying developers' needs and interaction styles affect the required support from a programming assistant?

> **RQII.3** – How can human-in-the-loop methods improve transparency and control of personalization while minimizing developer effort?

As mentioned earlier in this section, we will explore both implicit and explicit personalization approaches. Implicit personalization reduces effort for users, but creates a risk of profiling by making assumptions about the user [11]. Explicit personalization decreases

this risk through increased transparency and control, but also increases effort. This highlights the needs for carefully designed personalization mechanisms that balances these trade-offs.

Implicit personalization requires our programming assistant to be able to infer developers' needs from their interaction style. In Q **Study C**, we will assess if and how we can do this by exploring classical NLP techniques and LLM approaches on our interaction dataset from study **Study A**, answering $\mathbf{RQ_{II.1}}$. Since we risk increasing effort for specific groups of developers if this mechanism does not function properly [11], we will conduct careful testing to ensure our approach is not biased.

Next in Q **Study D**, we will employ a vignette survey. In **Study A** and **Study B** we already explore what aspects of interaction should be tailored to individual developers and contexts. For each combination of contextual factors, we will craft several vignettes: scenarios describing the context as well as alternative (non-personalized) ways in which an assistant might support the developer which participants will be asked to rate. By relating contextual factors and preferred forms of support to participants' needs and interaction style, which we obtain through a preliminary questionnaire, we answer $\mathbf{RQ_{II.2}}$. In this same study, we will also to explore several forms of human-in-the-loop approaches for explicit personalization, such as displaying and allowing edits to inferred needs, and prompting for clarification if a developer's needs are ambiguous, answering $\mathbf{RQ_{II.3}}$.

3.4 Phase III – Implementing a Personalized Programming Assistant Prototype

> $\mathbf{RQ_{III.1}}$ – How can personalized programming assistance be evaluated?

> $\mathbf{RQ_{III.2}}$ – How effectively does the proposed prototype personalize interactions?

In Q **Study E** [19], we argued how the large scale and high frequency of evaluation that LLM-based conversational assistants require, especially when employing personalization, makes it infeasible to evaluate them only using traditional (manual) human-centered approaches. In this position paper, we advocate combining insights from human-computer interaction (HCI) and artificial intelligence (AI) research to enable human-centered automatic evaluation of LLM-based conversational SE assistants to complement traditional evaluation. By employing LLMs as a simulated user of a programming assistant and through the LLM-as-a-Judge approach, researchers can potentially gain valuable insights aligning with those from actual user studies. Although not empirically answering $\mathbf{RQ_{III.1}}$, this paper lays the conceptual groundwork for future investigation by identifying requirements for such evaluation and challenges down the road.

In Q **Study F**, we will integrate all knowledge synthesized in the previous studies into a personalized programming assistant prototype to evaluate the effectiveness of our personalization approach. This approach encompasses implicit personalization through inference of the developers' needs, and explicit personalization by allowing developers to configure, inspect and edit their needs. We will leverage existing prompting techniques to adapt the assistants' support to developers' needs, guiding the LLM behind the assistant using our previous insights on how this support should be tailored. During the development phase of the prototype, we will implement and employ the automatic evaluation framework described in **Study E**, as well as conduct a pilot study with $n = 10$ participants with diverse personal profiles. The pilot study, as well as the later user study, will encompass a range of code-centric tasks including code maintenance, debugging, evolution, testing, and review. The results from the pilot study will be used to inform improvements to the assistant and the automatic evaluation framework. Specific focus will be placed on assessing the presence and extent of bias in the assistant and evaluation framework, which poses significant risks when employing personalization [11] and automated human-centered evaluation [19]. Finally, a user study with will be conducted, as well as a large-scale automatic evaluation of the final prototype using our evaluation framework. The results of the user study will be compared with the automatic evaluation to assess the framework's reliability, answering $\mathbf{RQ_{III.1}}$. Should the framework be shown to be sufficiently informative, reliable, and bias-free, its results will be used together with those of the user study to evaluate the personalized assistant, answering $\mathbf{RQ_{III.2}}$.

4 Risks and Mitigation Strategies

Conducting multiple studies on different aspects of the main research question creates a risk of theoretical drift. We attempt to mitigate this by basing the research in strong theoretical principles (see Section 2).

Conducting several user studies in addition to a case study increases the risk of overgeneralization. This is compounded by personal and contextual factors potentially interacting in complex intersectional ways. Although we limit the factors we address, we cannot fully disentangle these within the scope of this research. We mitigate the risk of overgeneralization by reporting on the diversity in our participant samples and clearly avoiding generalized assumptions about individual differences.

Personalization mechanisms themselves also run the risk of reinforcing bias and stereotyping when designed from overgeneralized insights. We address these concerns by employing human-in-the-loop feedback mechanisms and emphasizing user consent.

5 Relevance to CHASE Doctoral Symposium

As research into and adoption of LLMs in software engineering increases rapidly, the need for LLM-based tools to take into account human aspects increases. Since the the research line detailed in this paper is focused on tailoring conversational assistants in order to better support diversity in developers, we think the CHASE Doctoral Symposium provides an ideal opportunity to gain feedback and advice from the community. Besides welcoming any comments in general, we specifically invite the committee to share their views on 1) whether the research is sufficiently based in current literature and relevant theory, and 2) whether the expected contributions are ambitious enough and make for a strong doctoral thesis.

Acknowledgments

The author is advised by Prof. Dr. Frits Vaandrager and co-advised by Dr. Mairieli Wessel from Radboud University.

References

[1] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. 2021. Program Synthesis with Large Language Models. arXiv:2108.07732. arXiv:2108.07732 [cs]

[2] David Benyon. 2019. *Designing User Experience: A Guide to HCI, UX and Interaction Design* (fourth edition ed.). Pearson, Harlow New York Toronto.

[3] Margaret Burnett, Simone Stumpf, Jamie Macbeth, Stephann Makri, Laura Beckwith, Irwin Kwan, Anicia Peters, and William Jernigan. 2016. GenderMag: A Method for Evaluating Software's Gender Inclusiveness. *Interacting with Computers* 28, 6 (Nov. 2016), 760–787. doi:10.1093/iwc/iwv046

[4] Ruijia Cheng, Ruotong Wang, Thomas Zimmermann, and Denae Ford. 2024. "It Would Work for Me Too": How Online Communities Shape Software Developers' Trust in AI-powered Code Generation Tools. *ACM Transactions on Interactive Intelligent Systems* 14, 2, Article 11 (May 2024), 39 pages. doi:10.1145/3651990

[5] Rudrajit Choudhuri, Dylan Liu, Igor Steinmacher, Marco Gerosa, and Anita Sarma. 2024. How Far Are We? The Triumphs and Trials of Generative AI in Learning Software Engineering. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. ACM, Lisbon Portugal, 1–13. doi:10.1145/3597503.3639201

[6] Fiona Draxler, Daniel Buschek, Mikke Tavast, Perttu Hämäläinen, Albrecht Schmidt, Juhi Kulshrestha, and Robin Welsch. 2023. Gender, Age, and Technology Education Influence the Adoption and Appropriation of LLMs. arXiv:2310.06556. arXiv:2310.06556 [cs] doi:10.48550/arXiv.2310.06556

[7] Haiyan Fan and Marshall Scott Poole. 2006. What Is Personalization? Perspectives on the Design and Implementation of Personalization in Information Systems. *Journal of Organizational Computing and Electronic Commerce* 16, 3-4 (Jan. 2006), 179–202. doi:10.1080/10919392.2006.9681199

[8] Marc Hassenzahl. 2003. The Thing and I: Understanding the Relationship Between User and Product. In *Funology: From Usability to Enjoyment*, Mark A. Blythe, Andrew F. Monk, Kees Overbeeke, and Peter C. Wright (Eds.). Kluwers, Dordrecht, 31–42.

[9] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. 2024. Large Language Models for Software Engineering: A Systematic Literature Review. *ACM Trans. Softw. Eng. Methodol.* 33, 8, Article 220 (Dec. 2024), 79 pages. doi:10.1145/3695988

[10] Ranim Khojah, Mazen Mohamad, Philipp Leitner, and Francisco Gomes De Oliveira Neto. 2024. Beyond Code Generation: An Observational Study of ChatGPT Usage in Software Engineering Practice. *Proceedings of the ACM on Software Engineering* 1, FSE (July 2024), 1819–1840. doi:10.1145/3660788

[11] Hannah Rose Kirk, Bertie Vidgen, Paul Röttger, and Scott A. Hale. 2024. The Benefits, Risks and Bounds of Personalizing the Alignment of Large Language Models to Individuals. *Nature Machine Intelligence* 6, 4 (April 2024), 383–392. doi:10.1038/s42256-024-00820-y

[12] Stefano Lambiase, Gemma Catolino, Fabio Palomba, Filomena Ferrucci, and Daniel Russo. 2026. Investigating the Role of Cultural Values in Adopting Large Language Models for Software Engineering. *ACM Transactions on Software Engineering and Methodology* 35, 1 (Jan. 2026), 1–43. doi:10.1145/3725529

[13] Jenny T. Liang, Chenyang Yang, and Brad A. Myers. 2024. A Large-Scale Survey on the Usability of AI Programming Assistants: Successes and Challenges. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering (ICSE '24)*. Association for Computing Machinery, New York, NY, USA, 1–13. doi:10.1145/3597503.3608128

[14] Daye Nam, Andrew Macvean, Vincent Hellendoorn, Bogdan Vasilescu, and Brad Myers. 2024. Using an LLM to Help With Code Understanding. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. ACM, Lisbon Portugal, 1–13. doi:10.1145/3597503.3639187

[15] Sydney Nguyen, Hannah McLean Babe, Yangtian Zi, Arjun Guha, Carolyn Jane Anderson, and Molly Q Feldman. 2024. How Beginning Programmers and Code LLMs (Mis)Read Each Other. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*. ACM, Honolulu HI USA, 1–26. doi:10.1145/3613904.3642706

[16] Donald Norman. 1986. Cognitive Engineering. In *User Centered System Design: New Perspectives on Human-Computer Interaction*. CRC Press, 31–61. doi:10.1201/b15703-3

[17] Sebastian A. C. Perrig, Lena Fanya Aeschbach, Nicolas Scharowski, Nick von Felten, Klaus Opwis, and Florian Brühlmann. 2024. Measurement Practices in User Experience (UX) Research: A Systematic Quantitative Literature Review. *Frontiers in Computer Science* 6 (March 2024), 1368860. doi:10.3389/fcomp.2024.1368860

[18] Jonan Richards and Mairieli Wessel. 2024. What You Need Is What You Get: Theory of Mind for an LLM-Based Code Understanding Assistant. In *2024 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, Flagstaff, AZ, USA, 666–671. doi:10.1109/ICSME58944.2024.00070

[19] Jonan Richards and Mairieli Wessel. 2025. Bridging HCI and AI Research for the Evaluation of Conversational SE Assistants. arXiv:2502.07956. arXiv:2502.07956 [cs] doi:10.48550/arXiv.2502.07956

[20] Steven I. Ross, Fernando Martinez, Stephanie Houde, Michael Muller, and Justin D. Weisz. 2023. The Programmer's Assistant: Conversational Interaction with a Large Language Model for Software Development. In *Proceedings of the 28th International Conference on Intelligent User Interfaces*. ACM, Sydney NSW Australia, 491–514. doi:10.1145/3581641.3584037

[21] Daniel Russo. 2024. Navigating the Complexity of Generative AI Adoption in Software Engineering. *ACM Trans. Softw. Eng. Methodol.* 33, 5 (June 2024), 135:1–135:50. doi:10.1145/3652154

[22] Hariharan Subramonyam, Roy Pea, Christopher Lawrence Pondoc, Maneesh Agrawala, and Colleen Seifert. 2024. Bridging the Gulf of Envisioning: Cognitive Design Challenges in LLM Interfaces. arXiv:2309.14459. arXiv:2309.14459 [cs] doi:10.48550/arXiv.2309.14459

[23] Tao Xiao, Christoph Treude, Hideaki Hata, and Kenichi Matsumoto. 2024. DevGPT: Studying Developer-ChatGPT Conversations. In *Proceedings of the 21st International Conference on Mining Software Repositories (Msr '24)*. Association for Computing Machinery, New York, NY, USA, 227–230. doi:10.1145/3643991.3648400