In-IDE Human-AI Experience in the Era of Large Language Models
A Literature Review

Sergeyuk, Agnia; Titov, Sergey; Izadi, Maliheh

In-IDE Human-AI Experience in the Era of Large Language Models; A Literature Review

Agnia Sergeyuk
agnia.sergeyuk@jetbrains.com
JetBrains Research
Belgrade, Serbia

Sergey Titov
sergey.titov@jetbrains.com
JetBrains Research
Paphos, Cyprus

Maliheh Izadi
m.izadi@tudelft.nl
Delft University of Technology
Delft, Netherlands

ABSTRACT

Integrated Development Environments (IDEs) have become central to modern software development, especially with the integration of Artificial Intelligence (AI) to enhance programming efficiency and decision-making. The study of in-IDE Human-AI Experience is critical in understanding how these AI tools are transforming the software development process, impacting programmer productivity, and influencing code quality.

We conducted a literature review to study the current state of in-IDE Human-AI Experience research, bridging a gap in understanding the nuanced interactions between programmers and AI assistants within IDEs. By analyzing 36 selected papers, our study illustrates three primary research branches: **Design**, **Impact**, and **Quality** of Interaction.

The trends, challenges, and opportunities identified in this paper emphasize the evolving landscape of software development and inform future directions for research, and development in this dynamic field. Specifically, we invite the community to investigate three aspects of these interactions: designing **task-specific user interface**, building **trust**, and improving **readability**.

CCS CONCEPTS

• **Human-centered computing → Human computer interaction (HCI)**;

KEYWORDS

Human-Computer Interaction, Artificial Intelligence, Integrated Development Environment, Programming, User Studies, User Experience

1 INTRODUCTION

In the ever-evolving landscape of technology, Human-Computer Interaction (HCI) stands as a key field, encompassing the study of design, implementation, and usage of computer technology [27]. This dynamic discipline, bridging hardware and software, delves deep into understanding and optimizing the interaction between humans and digital systems. HCI not only investigates how users experience these technologies but also aims to enhance this interaction to be more intuitive, efficient, and user-centric.

However, the landscape of HCI has been significantly transformed with the advent and evolution of Large Language Models (LLMs) and advanced Artificial Intelligence (AI) assistance tools. This technological progression has shifted the focus from conventional HCI paradigms to a more nuanced and interactive domain, known as Human-AI Experience (HAX) [1]. HAX represents a deeper integration of AI within user interactions, where AI is not just a tool but a collaborative partner, reshaping the way users engage with and benefit from technological systems. This shift underscores the growing importance of understanding and designing for interactions that are increasingly AI-driven, marking a pivotal change in the field of Human-Computer Interaction.

With this change, Integrated Development Environments (IDEs) as the primary platform for Developer-AI interactions, are transforming beyond mere code creation tools into sophisticated ecosystems where AI's role extends from assisting in coding to offering intelligent insights and recommendations.

As a result, the focus of in-IDE HCI studies, initially concentrated on optimizing day-to-day programming activities from an HCI perspective, is now shifting towards in-IDE HAX, a more integrated and interactive approach. This transition reflects a growing interest in not just improving functional aspects of the developer's experience, but also in understanding and enhancing the collaborative dynamics between the developer and AI within the IDE.

As the field of in-IDE HAX evolves, there is a growing need for centralized knowledge and collaboration within the community. A collaborative approach can enhance efficiency, minimize redundant efforts, and encourage inspiration among researchers, contributing to the advancement of the field. In response to the absence of a literature survey in this domain, this paper offers an overview of existing in-IDE HAX research, distilling main directions and insights to guide future investigations.

Our literature survey was conducted with a focus on inclusivity and reliability. We selected prominent online databases known for their relevance and rigor in the fields of software engineering, AI, and HCI, specifically choosing the ACM Digital Library, DBLP, IEEE Digital Library, and ArXiv. The selection of ArXiv was particularly important due to the rapid development in our field. Our search was targeted at research articles published from 2020 onwards, aligning with the recent advancements in the field. This initial search yielded a pool of 211 papers. We then carefully applied a

set of criteria to ensure only papers directly relevant to in-IDE HAX were included. This filtering process led to the selection of 36 significant papers, published between 2020 and 2024, which collectively offer a comprehensive perspective for our study.

Analyzing the papers, we managed to identify three key research areas. The first group of studies [16, 20, 33, 38] focuses on integrating AI assistants into programming environments, emphasizing the impact of user interface and proposing design principles for user-friendly AI assistance. Second group [10, 26, 39] explores how AI assistance reshapes and impacts the programming workflow, categorizing interactions within it. The third category of studies [8, 24, 25] assesses AI assistant functionality, examining correctness, understandability, and security. Together, these studies give us important insights into the **Design, Impact, and Quality** of in-IDE HAX, guiding future research in this fast-changing area.

The recognition of design principles provides insights and indicates promising directions for future research, extending their application beyond conventional chat-based and code-completion formats. Understanding the shift in programmers' workflow emphasizes the importance of dedicated research to support users in adapting to these changes, especially in educational settings where accommodating workflow shifts significantly impacts the learning experience. Regarding quality, a clear trajectory for future investigations involves fine-tuning models to enhance security and correctness. This is particularly crucial for addressing non-trivial tasks and accommodating varied programming languages.

The present work not only consolidates the existing knowledge in the field of in-IDE HAX but also provides a foundation for future research efforts. The identified research areas, curated dataset, and proposed directions contribute to the collective understanding of the evolving dynamics between Humans and AI within IDEs.

2 METHOD

To understand the landscape of existing research in the field of in-IDE HAX, we conducted a literature survey with the following Research Questions: **"What research has already been done in this field"** and **"What are the dimensions of Human-AI Interaction inside IDE that should be addressed in the research aiming at supporting developers at their work"**.

We took the following steps to ensure the inclusivity and reliability of the literature review.

We first selected four well-known digital libraries to encompass a diverse range of scholarly sources. This includes ACM Digital Library, DBLP, IEEE Digital Library, and ArXiv. We intentionally included ArXiv in the list to account for emerging fields, where useful insights might be collected from not peer-reviewed papers.

Next, we employed a search strategy using the following search string to identify relevant research papers:

("Integrated Development Environment" OR "IDE" OR "VS code" OR "Intellij" OR "vim" OR "Code editor") AND ("AI assistant" OR "AI features" OR "Human-AI interaction" OR "Human-AI collaboration" OR "AI coding assistant" OR "LLM-based code generation" OR "AI code generation" OR "Conversational AI").

We then applied several inclusion and exclusion criteria to filter the retrieved literature. We excluded studies published before 2020,

Table 1: Statistics on study sources

Venue	Count
ACM on Programming Languages	1
ACM Transactions on Computer-Human Interaction	2
Conference on Human Factors in Computing Systems	1
International Conference on Software Engineering	2
International Conference on Mining Software Repositories	1
International Conference on Evaluation and Assessment in Software Engineering	1
International Conference on Intelligent User Interfaces	2
International Symposium on Machine Programming	1
Technical Symposium on Computer Science Education	1
Symposium on User Interface Software and Technology	2
ArXiv	22
Total	36

as well as blog posts. Additionally, we filtered out non-English manuscripts and those with titles, abstracts, or full texts that are thematically irrelevant ensuring the relevance and recency of the selected studies.

During the process, we initially collected 211 papers, with most of them (191) being sourced from ArXiv. After filtering based on the criteria, we extracted relevant information, including publication year, authorship, the study's overarching goal, research questions, methodology, and key findings from each selected source. This process resulted in 36 studies published in various conferences (11 studies), journals (3 studies), and finally yet only on ArXiv (22 studies). Table 1 Provides d breakdown of the number of studies per source.

3 FINDINGS

Our literature survey revealed notable insights into the integration of AI assistance in IDEs. It is clear that this integration goes beyond altering the user interface; it fundamentally changes developers' workflows and the core models of AI assistants. Consequently, as in-IDE HAX evolves, we are seeing the growth of three distinct research areas, each with its specific focus:

First is the exploration of User Interface **Design** in AI-enabled tools. This area seeks to answer key questions about design considerations when incorporating AI technologies into programming environments. Out of the 36 papers we reviewed, 14 specifically addressed this topic.

Second is the investigation of the **Impact** of HAX. This line of research centers on understanding how users interact with, perceive, and benefit from AI-based tools in different programming scenarios. It addresses common themes like usability issues, the effect on productivity, and user trust in these tools. Of the 36 papers we reviewed, 13 explored this aspect of the research field.

Third is the investigation of the **Quality** of AI Assistants. This research area delves into the performance of AI assistants, specifically looking at factors like correctness, understandability, security,

and their capability to solve algorithmic problems. Among the 36 papers we reviewed, 9 papers specifically focused on examining these aspects of AI assistants.

Studies of Design of Integration [3, 6, 9, 14, 16, 19, 20, 23, 28, 29, 33, 34, 37, 38] focus on prototyping integration of AI assistants into existing programming environments, systematizing methodologies and proposing principles for designing AI-assisted programming. Findings in this field provide evidence that the user interface of in-IDE AI assistance affects the usefulness of this tool and should be built thoughtfully.

The design principles for AI assistance, as detailed in the papers [16, 20, 34, 38], highlight the importance of clear communication and user control in generative AI, while code assistants and autocompletion features aim for adaptability, clarity, and user-friendly interactions.

Generative AI design principles involve communicating probabilistic nature and variable outputs, facilitating user annotation and visualization of differences, accommodating imperfection through feedback, implementing user-driven controls, providing sandbox environments, orienting users through mental models, and communicating capabilities and limitations, with a focus on preventing risks and harms [38]. Code assistants, following principles from [20], act as adaptable ghostwriters, providing ambient or interactive annotations, offering context control, balancing politeness and promotion, integrating search and documentation, shaping user expectations, and incorporating means of verification into code generation. Autocompletion features a user interface with suggestions near the input, allowing user decisions and leveraging partial input for predictions [16]. Autocompletion design principles include glanceable suggestions, juxtaposition for clarity, simplicity through familiarity, sufficient visibility for validation, and snoozability of suggestions to prevent interruptions [34].

Multiple papers delved into presenting AI as a pair programmer to assist novices in their education [19, 28], revealing a generally high level of acceptance and usefulness. Despite this, challenges persist in designing a user-friendly interface for such agents and need to be addressed.

The other set of papers investigates the **Design and Impact** of various AI integrations into IDEs. It delves into how diverse forms of interaction influence novices' workflows [14, 19, 28], the level of user trust in the assistant [23, 37], and user productivity [3, 6, 9, 29]. The results provided suggest that conversational and explainable AI might support user productivity and trust while, in some cases, causing overreliance.

Studies of the Impact of Interaction [2, 5, 10, 13, 15, 17, 21, 26, 35, 36, 39, 42, 43] aim to understand how programmers use and perceive AI assistants. The investigations also include identifying challenges faced by developers, creating a taxonomy of programmer activities involving AI, and assessing users' productivity, as well as their level of trust in these tools.

Research in this field shows that in-IDE Human-AI Interaction significantly affects and changes the developers' workflow.

Using AI tools increases productivity but may involve a trade-off in code quality since developers sometimes struggle to receive from AI the outputs that would align with their requirements and expectations [10, 13, 17, 35, 36, 39, 43]. The context in which AI tools are used, the quality of suggestions, and compatibility issues play crucial roles in shaping these tools' overall effectiveness and user perception.

The findings emphasize that interactions between programmers and AI assistants alter the traditional programming workflow. This transformation introduces dedicated time for interacting with AI and processing its outputs [21] and can be categorized into various modes of HAX [5, 26]. The findings suggest that successful AI integration involves balancing the benefits of acceleration with the need for careful exploration and validation.

In assessing the impact of AI assistance on novices, it becomes evident that while AI tools can positively influence programming education by facilitating learning and boosting motivation, challenges such as over-reliance and the need for cautious acceptance require attention [2, 15, 26]. These challenges might be addressed by implementing educational strategies and awareness programs.

Studies of Quality of Interaction [4, 8, 11, 22, 24, 25, 30, 40, 41] are qualitatively and quantitatively evaluating various aspects of AI assistants' functionality, ranging from correctness and understandability of suggested code to investigating security concerns.

These studies reveal that the effectiveness and utility of an AI assistant depend not only on its user interface but also on the quality of the model's output. Moreover, there is no one-size-fits-all solution for the core model choice, as variations can impact user productivity and the acceptance rate of suggestions.

Most of the works are focused on the correctness of LLM outputs, analyzing if the suggestions provided are relevant to the task and error-free [8, 11, 24, 30, 40, 41]. Findings suggest that while AI assistants can provide relevant solutions and suggestions, especially for trivial types of tasks, they still might be erroneous and require the user's ability to notice and correct them.

Regarding the comprehensibility and complexity of the code, several research show that the assistants generally produce understandable code that might be even less complex than one written by human [8, 24, 40].

A few studies are focused on security, assessing if the assistance introduces security vulnerabilities to the code and to what extent [13, 25, 30]. From these studies, it can be derived that security risks might depend on the model used for the assistance foundation since, in some cases, the rate of vulnerability was low, but in some cases relatively high, reaching 40% of generated programs on C language appeared to be vulnerable.

Some papers covered how fine-tuning the foundational model can help to improve the quality of interaction [11, 22, 41] — support accuracy and timing of the shown suggestions to enhance the acceptance rate.

Exploring the interplay between models' characteristics, user interfaces, and real-world usage can provide valuable insights into optimizing AI assistants for diverse programming tasks and improving developers' overall experience.

4 DISCUSSION

In examining the literature, three distinct categories emerge, each shedding light on various aspects of AI assistance.

The first category explores the **Design** of HAX, drawing insights from interviews, literature reviews, experiments, or experiences in

other domains. Some papers within this category focus solely on design, while others extend their work to investigate how different forms of AI impact users.

The identified design principles, including a seamless user experience, visibility of suggestions, and considerations for context control, snoozability, and potential harm mitigation, form a foundation for developing user-friendly AI assistance tools. The challenges associated with presenting AI as a pair programmer for novices underscore the need for ongoing efforts in designing interfaces that cater to diverse user levels, ensuring acceptance and usefulness.

The second category delves into the **Impact** of HAX, questioning whether its presence affects users and, if so, in what ways.

The integration of AI tools, while enhancing productivity, introduces a nuanced trade-off in code quality. Developers grapple with aligning AI-generated outputs with their requirements, highlighting the intricate interplay of contextual factors, suggestion quality, and compatibility issues. The identified modes of HAX and the necessity for careful exploration and validation underscore the need for a balanced approach, ensuring the positive acceleration of workflow without compromising code quality.

The third category centers on the **Quality** of HAX, probing into the ability of AI to provide secure, comprehensible, and adequate code suggestions.

The findings emphasize that the effectiveness of AI assistants hinges not only on the user interface but also on the quality of the underlying models. Notably, studies highlighted the variability in the correctness of LLM outputs, indicating a need for user vigilance. Comprehensibility and code complexity were identified as areas where AI assistants generally perform well. However, security assessments revealed potential vulnerabilities, underscoring the importance of fine-tuning foundational models to enhance overall interaction quality.

4.1 FUTURE WORK

Specifically, within our area of interest, we suggest focusing on three aspects: **Task-specific user interface** in the realm of Design of Interaction, **Trust** in Impact of Interaction, and **Readability** in Quality of Interaction.

In the domain of the **Design** of HAX, we propose that chat-based interaction with LLMs may not always be the most effective approach, and different tasks may require alternative methods. We suggest two complementary directions within this area: (1) Analyzing common challenges addressed through chat-based models and implementing them in more convenient non-chat formats. For instance, writing method documentation is efficiently accomplished through IDE plugins with context menu buttons. [1] (2) Enhancing model reactivity by transforming predictable actions into automatic suggestions presented to the user. For example, automatically proposing fixes after an error or suggesting the optimization after using the profiler. This enables a seamless and proactive AI interaction experience. While we believe chat-based interaction creates a lot of traction in Human-AI Interaction, it is an interesting challenge to try to improve inherently chat-based interactions —

---

[1]Documentation generation is a feature in AI-assistant plugin for IntelliJ IDEA: https://plugins.jetbrains.com/plugin/22282-ai-assistant

like question answering about the code base. Lastly, there is uncharted territory of the interfaces for the no-code and multi-agent development tools. What shape will the development process be if users interact with several AI assistants or even create software using only natural language without code itself?

By focusing on trust, we aim to account for the developers' attitudes to AI, which might **Impact** the interaction with technology. One possible approach to the trust problem is looking at the Natural Language Processing domain. Such approaches as highlighting the tokens that affected the output the most [18], approximating the uncertainty of the model [12] or just providing clear and transparent context which was provided to the model could be beneficial for facilitating trust between AI and the user. While such methods have already been developed, it is still a challenge to introduce them into the context of the IDE without hurting user productivity and the development process.

Regarding the **Quality** of interaction, we believe that readability is a promising concept for code models' alignment — we suggest that given equal correctness of different models' outputs, people would prefer the model that will produce more readable code since such outputs will improve users' productivity while working with it. While there are multiple methods of LLMs alignment developed [7], there are just a few models of code readability [32] and studies about how people discriminate between readable and non-readable code [31]. Therefore, it is an open question whether we can align a model to consistently produce more code that will be perceived as more readable by actual users. In this direction of work, we suggest looking for general properties of code that people find desirable to see, or at least some strata of programmers with a common view of what desirable code is, and providing for the specific fine-tuning.

4.2 THREATS TO VALIDITY

While gathered findings provide valuable insights, in any scientific study, it is essential to consider potential threats to validity, which are factors that can affect the accuracy and generalizability of the research findings.

Sampling Bias: Despite rigorous efforts to include well-known libraries and refine the search string, the possibility of sampling bias remains. While our approach aimed to minimize bias, the inherent challenge of capturing every relevant work persists. That is why we provide our search protocol to mitigate this threat.

Temporal Bias: The chosen timeframe (papers published between 2020 and 2024) introduces a potential temporal bias, excluding earlier works. This decision was driven by the intention to focus on contemporary developments following the advent of LLMs. While acknowledging potential temporal limitations, we aimed to capture the latest advancements and trends in this rapidly evolving field.

Source Reliability: Inclusion of non-peer-reviewed papers from ArXiv introduces concerns regarding the reliability of findings, given the absence of a formal peer-review process. Recognizing this limitation, we deemed it necessary to consider insights from ArXiv due to the dynamic and rapidly evolving nature of the field. Thoroughly investigating titles, abstracts, and full texts ensured that all included papers contributed meaningfully to our survey. Moreover, the open publication environment of ArXiv encourages

publishing negative or null results, contributing to a more balanced representation of research outcomes.

Interpretation Bias: Analyzing a large amount of information may introduce interpretation bias and impact how studies are categorized. While acknowledging this complexity, we emphasize transparency and provide the entire dataset. [2] for readers

5 CONCLUSION

The field of in-IDE Human-AI Experience has just started to develop, but some patterns and overall conclusions might be derived from the published works already. It is evident that the field has three branches, namely **Design, Impact, Quality** of Interaction. In the dimension of Design, research provide design principles for creating in-IDE AI assistance and making them usable, as well as exploring efficient options for AI integration into the educational process and IDEs of various kinds. The dimension of Impact mostly focuses on changes in developers' workflow and efficiency, as well as their overall experience with AI assistants. The dimension of Quality studies if the foundational models of AI assistants are correct enough, producing comprehensible and secure code, and if this quality can be increased by fine-tuning practices. Based on this discovery, we propose several specific research topics in each of the dimensions that might be covered by industrial research teams to address open questions in the field.

REFERENCES

[1] Saleema Amershi, Dan Weld, Mihaela Vorvoreanu, Adam Fourney, Besmira Nushi, Penny Collisson, Jina Suh, Shamsi Iqbal, Paul N. Bennett, Kori Inkpen, Jaime Teevan, Ruth Kikin-Gil, and Eric Horvitz. 2019. Guidelines for Human-AI Interaction. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) *(CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–13.
[2] Matin Amoozadeh, David Daniels, Daye Nam, Stella Chen, Michael Hilton, Sruti Srinivasa Ragavan, and Mohammad Amin Alipour. 2023. Trust in Generative AI among students: An Exploratory Study. arXiv:2310.04631 [cs.HC]
[3] Tyler Angert, Miroslav Suzara, Jenny Han, Christopher Pondoc, and Hariharan Subramonyam. 2023. Spellburst: A Node-based Interface for Exploratory Creative Coding with Natural Language Prompts. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology (UIST '23)*. ACM.
[4] Owura Asare, Meiyappan Nagappan, and N. Asokan. 2023. Is GitHub's Copilot as Bad as Humans at Introducing Vulnerabilities in Code? arXiv:2204.04741 [cs.SE]
[5] Shraddha Barke, Michael B. James, and Nadia Polikarpova. 2023. Grounded Copilot: How Programmers Interact with Code-Generating Models. *Proc. ACM Program. Lang.* 7, OOPSLA1, Article 78 (apr 2023), 27 pages.
[6] Yu Cheng, Jieshan Chen, Qing Huang, Zhenchang Xing, Xiwei Xu, and Qinghua Lu. 2023. Prompt Sapper: A LLM-Empowered Production Tool for Building AI Chains. arXiv:2306.12028 [cs.SE]
[7] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. 2017. Deep reinforcement learning from human preferences. *Advances in neural information processing systems* 30 (2017).
[8] Arghavan Moradi Dakhel, Vahid Majdinasab, Amin Nikanjam, Foutse Khomh, Michel C. Desmarais, Zhen Ming, and Jiang. 2023. GitHub Copilot AI pair programmer: Asset or Liability? arXiv:2206.15331 [cs.SE]
[9] Felicia Li Feng, Ryan Yen, Yuzhe You, Mingming Fan, Jian Zhao, and Zhicong Lu. 2023. CoPrompt: Supporting Prompt Sharing and Referring in Collaborative Natural Language Programming. arXiv:2310.09235 [cs.HC]
[10] Ken Gu, Madeleine Grunde-McLaughlin, Andrew M. McNutt, Jeffrey Heer, and Tim Althoff. 2023. How Do Data Analysts Respond to AI Assistance? A Wizard-of-Oz Study. arXiv:2309.10108 [cs.HC]
[11] Qing Huang, Dianshu Liao, Zhenchang Xing, Zhiqiang Yuan, Qinghua Lu, Xiwei Xu, and Jiaxing Lu. 2022. SE Factual Knowledge in Frozen Giant Code Model: A Study on FQN and its Retrieval. arXiv:2212.08221 [cs.SE]
[12] Yuheng Huang, Jiayang Song, Zhijie Wang, Shengming Zhao, Huaming Chen, Felix Juefei-Xu, and Lei Ma. 2023. Look Before You Leap: An Exploratory Study of Uncertainty Measurement for Large Language Models. arXiv:2307.10236 [cs.SE]
[13] Saki Imai. 2022. Is GitHub Copilot a Substitute for Human Pair-Programming? An Empirical Study. In *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings* (Pittsburgh, Pennsylvania) *(ICSE '22)*. Association for Computing Machinery, New York, NY, USA, 319–321.
[14] Dhanya Jayagopal, Justin Lubin, and Sarah E. Chasins. 2022. Exploring the Learnability of Program Synthesizers by Novice Programmers. In *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology* (Bend, OR, USA) *(UIST '22)*. Association for Computing Machinery, New York, NY, USA, Article 64, 15 pages.
[15] Majeed Kazemitabaar, Xinying Hou, Austin Henley, Barbara J. Ericson, David Weintrop, and Tovi Grossman. 2023. How Novices Use LLM-Based Code Generators to Solve CS1 Coding Tasks in a Self-Paced Learning Environment. arXiv:2309.14049 [cs.HC]
[16] Florian Lehmann and Daniel Buschek. 2020. Examining Autocompletion as a Basic Concept for Interaction with Generative AI. *i-com* 19, 3 (2020), 251–264.
[17] Jenny T. Liang, Chenyang Yang, and Brad A. Myers. 2023. A Large-Scale Survey on the Usability of AI Programming Assistants: Successes and Challenges. arXiv:2303.17125 [cs.SE]
[18] Scott M Lundberg and Su-In Lee. 2017. A Unified Approach to Interpreting Model Predictions. In *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.). Curran Associates, Inc., 4765–4774.
[19] Gilda Manfredi, Ugo Erra, and Gabriele Gilio. 2023. A Mixed Reality Approach for Innovative Pair Programming Education with a Conversational AI Virtual Avatar. In *Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering* (Oulu, Finland) *(EASE '23)*. Association for Computing Machinery, New York, NY, USA, 450–454.
[20] Andrew M. McNutt, Chenglong Wang, Robert A. DeLine, and Steven M. Drucker. 2023. On the Design of AI-powered Code Assistants for Notebooks. arXiv:2301.11178 [cs.HC]
[21] Hussein Mozannar, Gagan Bansal, Adam Fourney, and Eric Horvitz. 2023. Reading Between the Lines: Modeling User Behavior and Costs in AI-Assisted Programming. arXiv:2210.14306 [cs.SE]
[22] Hussein Mozannar, Gagan Bansal, Adam Fourney, and Eric Horvitz. 2023. When to Show a Suggestion? Integrating Human Feedback in AI-Assisted Programming. arXiv:2306.04930 [cs.HC]
[23] Daye Nam, Andrew Macvean, Vincent Hellendoorn, Bogdan Vasilescu, and Brad Myers. 2023. In-IDE Generation-based Information Support with a Large Language Model. arXiv:2307.08177 [cs.SE]
[24] Nhan Nguyen and Sarah Nadi. 2022. An Empirical Evaluation of GitHub Copilot's Code Suggestions. In *Proceedings of the 19th International Conference on Mining Software Repositories* (Pittsburgh, Pennsylvania) *(MSR '22)*. Association for Computing Machinery, New York, NY, USA, 1–5.
[25] Hammond Pearce, Baleegh Ahmad, Benjamin Tan, Brendan Dolan-Gavitt, and Ramesh Karri. 2021. Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions. arXiv:2108.09293 [cs.CR]
[26] James Prather, Brent N. Reeves, Paul Denny, Brett A. Becker, Juho Leinonen, Andrew Luxton-Reilly, Garrett Powell, James Finnie-Ansley, and Eddie Antonio Santos. 2023. "It's Weird That It Knows What I Want": Usability and Interactions with Copilot for Novice Programmers. 31, 1, Article 4 (nov 2023), 31 pages.
[27] Amon Rapp. 2023. Human–Computer Interaction. In *Oxford Research Encyclopedia of Psychology*.
[28] Peter Robe and Sandeep Kaur Kuttal. 2022. Designing PairBuddy—A Conversational Agent for Pair Programming. *ACM Trans. Comput.-Hum. Interact.* 29, 4, Article 34 (may 2022), 44 pages.
[29] 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* (Sydney, NSW, Australia) *(IUI '23)*. Association for Computing Machinery, New York, NY, USA, 491–514.
[30] Gustavo Sandoval, Hammond Pearce, Teo Nys, Ramesh Karri, Siddharth Garg, and Brendan Dolan-Gavitt. 2023. Lost at C: A User Study on the Security Implications of Large Language Model Code Assistants. arXiv:2208.09727 [cs.CR]
[31] Simone Scalabrino, Gabriele Bavota, Christopher Vendome, Mario Linares-Vásquez, Denys Poshyvanyk, and Rocco Oliveto. 2017. Automatically assessing code understandability: How far are we?. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 417–427.
[32] Simone Scalabrino, Mario Linares-Vásquez, Rocco Oliveto, and Denys Poshyvanyk. 2018. A comprehensive model for code readability. *Journal of Software: Evolution and Process* 30, 6 (2018), e1958.
[33] Chee Wei Tan, Shangxin Guo, Man Fai Wong, and Ching Nam Hang. 2023. Copilot for Xcode: Exploring AI-Assisted Programming by Prompting Cloud-based Large Language Models. arXiv:2307.14349 [cs.SE]
[34] Priyan Vaithilingam, Elena L. Glassman, Peter Groenwegen, Sumit Gulwani, Austin Z. Henley, Rohan Malpani, David Pugh, Arjun Radhakrishna, Gustavo Soares, Joey Wang, and Aaron Yim. 2023. Towards More Effective AI-Assisted Programming: A Systematic Design Exploration to Improve Visual Studio IntelliCode's User Experience. In *Proceedings of the 45th International Conference on*

---

[2]Dataset: https://doi.org/10.5281/zenodo.10290921

Software Engineering: Software Engineering in Practice (Australia) (ICSE-SEIP '23). IEEE Press, 185–195.

[35] Priyan Vaithilingam, Tianyi Zhang, and Elena L. Glassman. 2022. Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models. In Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems. ACM, 1–7.

[36] Mircea-Serban Vasiliniuc and Adrian Groza. 2023. Case Study: Using AI-Assisted Code Generation In Mobile Teams. arXiv:2308.04736 [cs.SE]

[37] Ruotong Wang, Ruijia Cheng, Denae Ford, and Thomas Zimmermann. 2023. Investigating and Designing for Trust in AI-powered Code Generation Tools. arXiv:2305.11248 [cs.HC]

[38] Justin D. Weisz, Michael Muller, Jessica He, and Stephanie Houde. 2023. Toward General Design Principles for Generative AI Applications. arXiv:2301.05578 [cs.HC]

[39] Justin D. Weisz, Michael Muller, Steven I. Ross, Fernando Martinez, Stephanie Houde, Mayank Agarwal, Kartik Talamadupula, and John T. Richards. 2022. Better Together? An Evaluation of AI-Supported Code Translation. In 27th International Conference on Intelligent User Interfaces (Helsinki, Finland) (IUI '22). Association

for Computing Machinery, New York, NY, USA, 369–391.

[40] Michel Wermelinger. 2023. Using GitHub Copilot to Solve Simple Programming Problems. In Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1 (Toronto ON, Canada) (SIGCSE 2023). Association for Computing Machinery, New York, NY, USA, 172–178.

[41] Wen Zhou, Seohyun Kim, Vijayaraghavan Murali, and Gareth Ari Aye. 2022. Improving Code Autocompletion with Transfer Learning. In Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice (Pittsburgh, Pennsylvania) (ICSE-SEIP '22). Association for Computing Machinery, New York, NY, USA, 161–162.

[42] Xiyu Zhou, Peng Liang, Beiqi Zhang, Zengyang Li, Aakash Ahmad, Mojtaba Shahin, and Muhammad Waseem. 2023. On the Concerns of Developers When Using GitHub Copilot. arXiv:2311.01020 [cs.SE]

[43] Albert Ziegler, Eirini Kalliamvakou, X. Alice Li, Andrew Rice, Devon Rifkin, Shawn Simister, Ganesh Sittampalam, and Edward Aftandilian. 2022. Productivity Assessment of Neural Code Completion. In Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming (San Diego, CA, USA) (MAPS 2022). Association for Computing Machinery, New York, NY, USA, 21–29.