

How Developers Interact with AI: A Taxonomy of Human-AI Collaboration in Software Engineering

Christoph Treude

*School of Computing and Information Systems
Singapore Management University
Singapore
ctreude@smu.edu.sg*

Marco A. Gerosa

*School of Informatics, Computing, and Cyber Systems
Northern Arizona University
Flagstaff, AZ, United States
marco.gerosa@nau.edu*

Abstract—Artificial intelligence (AI), including large language models and generative AI, is emerging as a significant force in software development, offering developers powerful tools that span the entire development lifecycle. Although software engineering research has extensively studied AI tools in software development, the specific types of interactions between developers and these AI-powered tools have only recently begun to receive attention. Understanding and improving these interactions has the potential to enhance productivity, trust, and efficiency in AI-driven workflows. In this paper, we propose a taxonomy of interaction types between developers and AI tools, identifying eleven distinct interaction types, such as auto-complete code suggestions, command-driven actions, and conversational assistance. Building on this taxonomy, we outline a research agenda focused on optimizing AI interactions, improving developer control, and addressing trust and usability challenges in AI-assisted development. By establishing a structured foundation for studying developer-AI interactions, this paper aims to stimulate research on creating more effective, adaptive AI tools for software development.

Index Terms—Artificial Intelligence, Software Development, Developer Tools, Human-AI Interaction, Generative AI, Large Language Models

I. INTRODUCTION

Artificial intelligence (AI) is rapidly transforming software development, bringing new capabilities and efficiency to every stage of the development lifecycle [1]. As AI tools become increasingly sophisticated, they offer developers a diverse range of support options, from autocompletion and code generation to documentation, testing, and project management [2]. Related work on developer interactions with traditional tools has established the importance of usability and integration [3], but there is limited research specifically on AI-powered tools [4]. The paper aims to address this gap.

This emerging focus on AI-powered tools brings with it an opportunity to explore how the interaction of the tools can be tailored to better align with developers' needs, ultimately enhancing adoption, trust, and productivity [5]. The diversity of interaction types – from auto-complete code suggestions to conversational exchanges [6] – introduces significant complexity in designing and evaluating these tools. Without a structured framework for analyzing these interactions, researchers and tool designers lack a common vocabulary and conceptual model for developer-AI interfaces. This fragmentation hinders

efforts to optimize how developers engage with AI tools, identify best practices, understand usage patterns, and address common challenges across different tools and contexts. A comprehensive taxonomy of developer-AI interactions can help bridge this gap by providing a foundation for empirical studies, enabling systematic tool evaluation, and guiding the development of AI assistance features. Furthermore, as AI capabilities continue to evolve rapidly, such a taxonomy becomes essential for tracking how interaction patterns adapt and emerge over time, ensuring that new tools and features align with developers' actual needs and workflows.

In this paper, we propose a taxonomy of interaction types between human developers and AI tools. To the best of our knowledge, this is the first taxonomy to specifically examine human-AI interaction types within the context of software engineering, a domain characterized by the need to manage diverse types of artifacts (e.g., code, bug reports, pull requests), navigate complex human collaboration dynamics, and address rapidly evolving development workflows. Building on this taxonomy, we outline a research agenda that highlights areas for further exploration, including optimizing interaction styles, improving developer control, and addressing trust and usability in AI-driven development. This paper aims to catalyze further research and discussion on creating more effective and adaptive AI tools for software development.

II. INTERACTION TYPES

Table I provides an overview of the 11 proposed types of developer-AI interactions, characterized by their triggers, the AI response, the developer reaction, the type of output generated, and concrete examples. The *trigger* column describes how each interaction is initiated, ranging from automatic mechanisms such as typing context to explicit commands or workflow events. The *AI response* column details the nature of the system's output, such as real-time suggestions, context-specific recommendations, or step-by-step explanations. The *developer response* column captures how developers interact with and react to AI output, including actions such as reviewing, refining, or integrating suggestions. The *output* column highlights the type of deliverables produced by AI, such as code snippets, documentation, or quality reports, offering insight into the specific contributions these interactions make

TABLE I
CHARACTERIZATION OF DEVELOPER-AI INTERACTION TYPES

Type	Trigger and AI Response	Developer Response and Output	Example
Auto-Complete Code Suggestions	<i>Trigger:</i> Automatic based on typing context. <i>AI Response:</i> Suggestions appear as ghost text or pop-ups.	<i>Developer Response:</i> Accept, scroll, or dismiss. <i>Output:</i> Suggestions.	Typing "def calculate_area" prompts a function body suggestion in GitHub Copilot [7].
Command-Driven Actions	<i>Trigger:</i> Explicit command input (e.g., <code>copilot:summary</code>). <i>AI Response:</i> Generates specified output (e.g., summary or documentation).	<i>Developer Response:</i> Review, edit, finalize. <i>Output:</i> Actions.	Using <code>copilot:summary</code> generates a pull request summary in GitHub [8].
Conversational Assistance	<i>Trigger:</i> Question or issue posed in a chat interface. <i>AI Response:</i> Step-by-step guidance, explanations, or snippets.	<i>Developer Response:</i> Copy, adapt, or ask follow-ups. <i>Output:</i> Explanations.	ChatGPT explains sorting a list of dictionaries [9].
Contextual Recommendations	<i>Trigger:</i> Interprets contextual cues (e.g., file type). <i>AI Response:</i> Suggests libraries, patterns, or improvements.	<i>Developer Response:</i> Evaluate, accept, modify. <i>Output:</i> Suggestions.	Sourcegraph Cody suggests database libraries for relevant files [10].
Selection-Based Enhancements	<i>Trigger:</i> Highlighting specific code segments. <i>AI Response:</i> Provides refactored code, explanations, or tests.	<i>Developer Response:</i> Review, incorporate, modify. <i>Output:</i> Actions.	Sourcery refactors highlighted functions [11].
Explicit UI Actions	<i>Trigger:</i> Button or icon clicked in the IDE. <i>AI Response:</i> Displays flagged issues, reports, or documentation.	<i>Developer Response:</i> Review, refine, incorporate. <i>Output:</i> Actions.	Security scans in IDEs flag vulnerabilities for review [12].
Comment-Guided Prompts	<i>Trigger:</i> Descriptive comments written by the developer. <i>AI Response:</i> Generates code beneath the comment.	<i>Developer Response:</i> Review, adjust, verify. <i>Output:</i> Actions.	Writing "/* Convert list of strings to uppercase */" prompts code in GitHub Copilot [7].
Event-Based Triggers	<i>Trigger:</i> Workflow events (e.g., commits or pull requests). <i>AI Response:</i> Reports issues or performs checks.	<i>Developer Response:</i> Review and address identified issues. <i>Output:</i> Actions.	GitLab Auto DevOps scans pull requests for vulnerabilities [13].
Shortcut-Activated Commands	<i>Trigger:</i> Shortcut keys pressed. <i>AI Response:</i> Provides suggestions or documentation in an overlay.	<i>Developer Response:</i> Evaluate, integrate, or dismiss. <i>Output:</i> Suggestions.	Shortcuts in IntelliJ IDEA open Copilot suggestions [14].
File-Aware Suggestions	<i>Trigger:</i> Recognizes file type or directory context. <i>AI Response:</i> Suggests templates or configuration options.	<i>Developer Response:</i> Review, accept, or adapt. <i>Output:</i> Suggestions.	CodeWhisperer suggests test templates for <code>.test</code> files [15].
Automated API Responses	<i>Trigger:</i> API calls or webhooks triggered by events. <i>AI Response:</i> Provides reports or release notes.	<i>Developer Response:</i> Review, modify, and integrate. <i>Output:</i> Reports.	CodeClimate analyzes pull requests and reports quality issues [16].

to the development process. The *example* column illustrates each interaction type with real-world applications, such as GitHub Copilot and ChatGPT. In the following, we briefly introduce each interaction type.

a) *Auto-Complete Code Suggestions:* Auto-complete code suggestions in tools such as GitHub Copilot represent one of the most intuitive ways developers interact with AI, as these suggestions seamlessly integrate into the development workflow. Triggered automatically based on typing context, AI offers real-time recommendations in the form of ghost text or pop-ups, reducing the cognitive effort required for repetitive tasks. Developers can easily accept, scroll through, or dismiss these suggestions, allowing them to maintain focus while improving productivity.

b) *Command-Driven Actions:* Command-driven actions allow developers to perform targeted tasks by issuing explicit instructions to the AI. These commands, such as generating documentation or summarizing code changes, empower

developers to quickly access specific functionality. The AI processes these directives to produce customized outputs that the developers can review, edit, and refine.

c) *Conversational Assistance:* Conversational assistance enables developers to engage in natural language interactions with AI, making it an approachable collaborator. Developers can ask questions or request guidance on specific challenges, and the AI responds with explanations, suggestions, or even detailed code snippets. This interactive exchange facilitates learning, problem solving, and creativity. By mimicking human-like communication, this type of interaction reduces the cognitive burden of translating ideas into formal syntax, allowing developers to focus more on their core problem-solving objectives rather than wrestling with technical documentation or rigid query formats.

d) *Contextual Recommendations:* Contextual recommendations leverage the AI's ability to interpret cues from the project environment, providing developers with tailored sug-

gestions that align with the specific context of their work. For example, the AI might suggest libraries, patterns, or improvements based on the file type, project dependencies, or overall structure. Unlike auto-complete code suggestions, which are triggered in real-time as the developer types and are focused on immediate, localized code completion, contextual recommendations take a broader view, offering guidance that aligns with the entire project or file. By focusing on project-wide context rather than line-by-line interactions, contextual recommendations complement autocomplete suggestions to provide more comprehensive support. Developers evaluate and integrate these suggestions as needed, ensuring that they align with the specific requirements of the task at hand.

e) Selection-Based Enhancements: Selection-based enhancements focus on specific portions of code, allowing developers to receive customized assistance for highlighted segments. The AI responds by providing refactored code, generating test cases, or offering explanations for complex logic. Developers can then incorporate or adjust these enhancements, improving code quality and clarity. By narrowing its focus to specific selections, the AI ensures that its responses are relevant and actionable, streamlining tasks such as debugging and optimization.

f) Explicit UI Actions: Explicit UI actions involve a more deliberate interaction, where developers manually trigger AI functionality through the tool interface. This might include running security scans, generating reports, or drafting documentation. The AI produces actionable outputs that developers can review, refine, and integrate into their workflow. This type of interaction emphasizes the importance of developer control, ensuring that AI support aligns precisely with their intentions.

g) Comment-Guided Prompts: Comment-guided prompts allow developers to influence AI behavior using natural language descriptions embedded directly in the code as comments. The AI interprets these prompts to generate code snippets that align with the described intent. Developers can then review, adjust, and verify the generated output to ensure that it meets their requirements.

h) Event-Based Triggers: Event-based triggers enable automation during key workflow milestones, such as commits or pull requests. The AI responds to these events by performing predefined tasks such as running quality checks or security scans and producing actionable outputs such as reports or flagged issues. Developers then review these outputs to address any identified concerns. This interaction type is used, for example, to integrate AI into continuous integration and delivery pipelines, automating repetitive tasks and ensuring code quality.

i) Shortcut-Activated Commands: Shortcut-activated commands streamline the interaction process by allowing developers to invoke AI functionality with predefined keyboard shortcuts. The AI responds with contextually relevant suggestions or documentation that developers can evaluate and incorporate into their work. This approach reduces disruption, allowing developers to maintain their workflow while accessing AI support quickly and efficiently.

In contrast, explicit UI actions require more deliberate engagement through IDE buttons or menus.

j) File-Aware Suggestions: File-aware suggestions provide developers with context-specific recommendations tailored to the type of file or directory they are working on. For example, the AI might suggest configuration options for a settings file or test templates for a testing script. Developers can review these suggestions and adapt them to fit their project requirements, ensuring that AI assistance is relevant and actionable. Note that file-aware suggestions provide recommendations tailored to specific file types at the time of file creation or editing. In contrast, contextual recommendations focus on broader project-level cues and suggest improvements during ongoing development.

k) Automated API Responses: Automated API responses integrate AI capabilities into broader project workflows through API calls or webhooks. These interactions typically involve the generation of reports, release notes, or analysis outputs in response to predefined triggers. Developers review, modify, and integrate these outputs as needed, making this interaction type particularly useful for managing large-scale or repetitive tasks within collaborative projects. Automated API responses focus on system-to-system interactions triggered by webhooks or API calls, whereas event-based triggers are specific to workflow events like commits or pull requests within the developer's environment.

III. RESEARCH AGENDA

The taxonomy presented in Section II provides a structured framework for understanding how developers interact with AI tools, serving as a foundation for identifying key research opportunities. Each type of interaction presents distinct challenges and areas for improvement. For instance, passive interactions such as code suggestions (e.g., GitHub Copilot) highlight questions about reducing cognitive load and balancing automation with developer control. Active interactions, such as conversational assistance or selection-based enhancements, raise additional considerations about trust, usability, and adaptability. These challenges and considerations form the basis for the research questions outlined in this agenda, which aim to align theoretical insights with practical advancements in developer-AI workflows.

Building on the taxonomy of developer-AI interaction types, this research agenda explores opportunities to optimize, expand, and tailor these interactions to better support developers. It highlights critical areas of study aimed at improving productivity, trust, customization, and usability in AI-driven software development tools.

a) Effectiveness of Interaction Types: Different interaction types, such as auto-triggered suggestions, command-driven actions, and selection-based enhancements, have distinct impacts on productivity and code quality. Some types may better suit novice developers, while others might benefit experienced professionals. To explore these differences, we need to understand: Which interaction types lead to the most productive workflows in specific software development tasks?

Are some types more suited to particular activities or user profiles? Comparative user studies that measure task completion times, accuracy, and satisfaction can help answer these questions. In addition, quantitative insights and qualitative feedback will reveal which types feel intuitive and effective in various scenarios.

b) Developer Trust and Adoption: Trust is a crucial factor for developers when adopting AI tools, influenced by the transparency, reliability, and level of control they offer. Developers may approach high-stakes tasks, such as security scans, with skepticism, often requiring clear and reliable outputs to build confidence in the tool's capabilities. One question to investigate is how different types of interaction shape trust in AI outputs. Another is how interaction designs can foster trust without encouraging over-reliance or distrust. Surveys and interviews can uncover trust perceptions, especially in scenarios where AI modifies code. Behavioral experiments could track how often developers accept or override AI suggestions, offering insight into trust-related behaviors.

c) Context-aware AI Interactions: AI tools often lack the ability to fully align their recommendations with the specific structure, goals, or context of a project. Incorporating project-level data, understanding dependencies, and recognizing design patterns could make AI suggestions more relevant and actionable. How could AI better adapt to the current task or the broader context of the project? Could integrating past interactions and project structure improve the relevance of suggestions? Developing context-aware models that embed these elements and comparing their acceptance rates with generic suggestions could provide valuable information.

d) Optimizing for Developer Control and Customization: Striking the right balance between automation and developer control is vital. Features that allow developers to adjust the behavior of AI tools, such as the frequency or intrusiveness of suggestions, can help ensure that the tools remain helpful rather than overwhelming. Research in this area should consider how interaction designs empower developers without introducing unnecessary complexity. Questions such as "What levels of customization do developers find most useful?" and "How can these features be implemented to enhance workflows?" can guide this work. Testing customizable interfaces and surveying developers about their preferences can help refine these designs.

e) Reducing Cognitive Load: AI tools can disrupt focus if their interactions are too frequent or intrusive. Minimizing cognitive load is essential for sustained productivity and engagement. A critical question is: What type of interaction best balances productivity and reduced cognitive burden? Research could compare passive or reactive interaction types to more intrusive ones. Methods such as eye tracking, task switching analysis, and subjective mental load assessments could help identify the least disruptive and most effective approaches.

f) Ethics and Bias in AI Interactions: Ethical concerns and biases in AI interactions must be addressed to ensure fair and inclusive development practices. AI tools risk reinforcing stereotypes or perpetuating biases present in their training

data. Investigating how AI suggestions differ according to developer characteristics or codebase patterns is an important step. What mechanisms can mitigate biases in AI outputs? Analyzing patterns across diverse developers and codebases can uncover problematic trends, while feedback systems that allow developers to flag issues can help refine AI behavior over time.

g) Privacy and Data Protection: One critical area for future research is understanding and mitigating privacy and data protection concerns in developer-AI interactions. AI-powered tools often rely on sending contextual data, such as code snippets or project metadata, to external servers for processing. This raises questions about how much of a developer's context is shared, who has access to the data, and how securely it is stored. Investigating privacy-preserving approaches, such as federated learning or on-device processing, could reduce the risks associated with exposing sensitive data. Future studies should evaluate trade-offs between the effectiveness of AI assistance and the privacy of developer workflows, exploring how these tools can operate without compromising security or confidentiality.

h) Hallucination and Damage Control: Hallucinations in AI, where the system generates incorrect or misleading outputs, pose a significant challenge in developer workflows, especially when such outputs are integrated into critical systems. Research is needed to identify the causes of these hallucinations and to develop mechanisms for detecting and mitigating them. For example, integrating AI systems with feedback loops that allow developers to report inaccurate suggestions could improve the reliability of these tools over time. Furthermore, establishing clear safeguards, such as confidence scores or requiring explicit developer review before accepting high-risk changes, can minimize potential damage caused by erroneous AI outputs.

IV. CONCLUSION

This paper presents a taxonomy of the types of developer-AI interaction, providing a structured framework for understanding the diverse ways developers engage with AI-powered tools in software engineering. The accompanying research agenda builds on this taxonomy, identifying critical areas such as trust, cognitive load, and customization that warrant further exploration. Our work provides a foundation for understanding the diverse ways developers interact with AI-powered tools in software engineering. The taxonomy outlines key interaction types and dimensions, which can help guide further research and tool development, e.g., to better orchestrate various interaction types throughout the development workflow. Although validation is beyond the scope of this paper, future studies could empirically evaluate the taxonomy by observing tool use in real-world software development workflows. Future work could also include a systematic evaluation of existing tools to analyze the frequency and usage of each interaction type. In addition, researchers could explore how these interaction types align with developer needs in specific contexts, refining the taxonomy based on empirical findings.

REFERENCES

- [1] D. Russo, “Navigating the complexity of generative AI adoption in software engineering,” *ACM Transactions on Software Engineering and Methodology*, 2024.
- [2] U. K. Durrani, M. Akpinar, M. F. Adak, A. T. Kabakus, M. M. Ozturk, and M. Saleh, “A decade of progress: A systematic literature review on the integration of AI in software engineering phases and activities (2013–2023),” *IEEE Access*, 2024.
- [3] R. Minelli, A. Moccia, M. Lanza, and L. Baracchi, “Visualizing developer interactions,” in *Proceedings of the IEEE Working Conference on Software Visualization*. IEEE, 2014, pp. 147–156.
- [4] A. Brown, S. D’Angelo, A. Murillo, C. Jaspan, and C. Green, “Identifying the factors that influence trust in AI code completion,” in *Proceedings of the ACM International Conference on AI-Powered Software*, 2024, pp. 1–9.
- [5] A. Murillo, A. Elizondo, S. D’Angelo, A. Brown, U. Kumar, Q. Madison, and A. Macvean, “Understanding and designing for trust in AI-powered developer tooling,” *IEEE Software*, vol. 41, no. 6, pp. 23–28, 2024.
- [6] S. I. Ross, F. Martinez, S. Houde, M. Muller, and J. D. Weisz, “The programmer’s assistant: Conversational interaction with a large language model for software development,” in *Proceedings of the International Conference on Intelligent User Interfaces*, 2023, pp. 491–514.
- [7] C. Bird, D. Ford, T. Zimmermann, N. Forsgren, E. Kalliamvakou, T. Lowdermilk, and I. Gazit, “Taking flight with Copilot: Early insights and opportunities of AI-powered pair-programming tools,” *ACM Queue*, vol. 20, no. 6, pp. 35–57, 2022.
- [8] T. Xiao, H. Hata, C. Treude, and K. Matsumoto, “Generative AI for pull request descriptions: Adoption, impact, and developer interventions,” *Proceedings of the ACM on Software Engineering*, vol. 1, no. FSE, pp. 1043–1065, 2024.
- [9] T. Xiao, C. Treude, H. Hata, and K. Matsumoto, “DevGPT: Studying developer-ChatGPT conversations,” in *Proceedings of the IEEE/ACM International Conference on Mining Software Repositories*. IEEE, 2024, pp. 227–230.
- [10] J. Hartman, H. Sagtani, J. Tibshirani, and R. Mehrotra, “AI-assisted coding with Cody: Lessons from context retrieval and evaluation for code recommendations,” in *Proceedings of the ACM Conference on Recommender Systems*, 2024, pp. 748–750.
- [11] A. Dwivedi, “More on AI tools: Developer’s magic wand,” in *Code-Mosaic: Learn AI-Driven Development and Modern Best Practices for Enterprise*. Springer, 2024, pp. 505–535.
- [12] R. Pudari and N. A. Ernst, “From Copilot to pilot: Towards AI supported software development,” *arXiv preprint arXiv:2303.04142*, 2023.
- [13] J. Shi, Z. Yang, H. J. Kang, B. Xu, J. He, and D. Lo, “Greening large language models of code,” in *Proceedings of the International Conference on Software Engineering: Software Engineering in Society*, 2024, pp. 142–153.
- [14] B. Zhang, P. Liang, X. Zhou, A. Ahmad, and M. Waseem, “Demystifying practices, challenges and expected features of using GitHub Copilot,” *International Journal of Software Engineering and Knowledge Engineering*, vol. 33, no. 11n12, pp. 1653–1672, 2023.
- [15] B. Mihaljević, A. Radovan, and M. Žagar, “An analysis of generative artificial intelligence tools usage to adapt and enrich software development courses,” in *Proceedings of the International Conference on Education and New Developments*, 2024, pp. 553–557.
- [16] Z. Hu and E. F. Gehringer, “Improving feedback on GitHub pull requests: A bots approach,” in *Proceedings of the IEEE Frontiers in Education Conference*. IEEE, 2019, pp. 1–9.