EXTENDED-ABSTRACT

Examining the Use and Impact of an AI Code Assistant on Developer Productivity and Experience in the Enterprise

JUSTIN D WEISZ, IBM Research, Yorktown Heights, NY, United States

Human-centered AI (HCAI) research leader focused on designing useful, safe, and personal AI systems that amplify and augment human capabilities. ACM Senior Member with numerous top-tier HCI & AI publications spanning human-AI collaboration & co-creativity, human-centered explainable AI (HCXAI), and UX design of AI applications. Uses qualitative, quantitative, prototyping, crowdsourcing, and speculative design methods to understand how to build effective AI user experiences.

SHRADDHA VIJAY KUMAR, Cisco Systems, Inc., India, Bengaluru, KA, India

MICHAEL MULLER, IBM Research, Yorktown Heights, NY, United States

KAREN ELLEN BROWNE, IBM Ireland Limited, Dublin, Ireland

ARIELLE GOLDBERG, International Business Machines, Armonk, NY, United States

KATRIN ELLICE HEINTZE, IBM Deutschland GmbH, Ehningen, Germany

View all

Open Access Support provided by:

IBM Research

IBM Ireland Limited

Cisco Systems, Inc., India

IBM Deutschland GmbH

IBM India Pvt Ltd

International Business Machines

Published: 26 April 2025

Citation in BibTeX format

CHI EA '25: Extended Abstracts of the CHI Conference on Human Factors in Computing Systems
April 26 - May 1, 2025
Yokohama, Japan

Conference Sponsors:
SIGCHI

Examining the Use and Impact of an AI Code Assistant on Developer Productivity and Experience in the Enterprise

Justin D. Weisz
IBM Research
Yorktown Heights, New York, USA
jweisz@us.ibm.com

Shraddha Vijay Kumar*
Cisco Systems, Inc.
Bangalore, India
kumarshraddha98@gmail.com

Michael Muller
IBM Research
Cambridge, Massachusetts, USA
michael_muller@us.ibm.com

Karen-Ellen Browne
IBM
Dublin, Ireland
karen-ellen@ibm.com

Arielle Goldberg
IBM
Poughkeepsie, New York, USA
arielle.goldberg1@ibm.com

Katrin Ellice Heintze
IBM
Boeblingen, Germany
ke.heintze@de.ibm.com

Shagun Bajpai
IBM
Kochi, India
shagun.bajpai@ibm.com

Abstract

AI assistants are being created to help software engineers conduct a variety of coding-related tasks, such as writing, documenting, and testing code. We describe the use of the watsonx Code Assistant (WCA), an LLM-powered coding assistant deployed internally within IBM. Through surveys of two user cohorts (N=669) and unmoderated usability testing (N=15), we examined developers' experiences with WCA and its impact on their productivity. We learned about their motivations for using (or not using) WCA, we examined their expectations of its speed and quality, and we identified new considerations regarding ownership of and responsibility for generated code. Our case study characterizes the impact of an LLM-powered assistant on developers' perceptions of productivity and it shows that although such tools do often provide net productivity increases, these benefits may not always be experienced by all users.

CCS Concepts

• **Human-centered computing** → **Empirical studies in HCI**; *Field studies*; • **Software and its engineering** → **Collaboration in software development**; *Automatic programming*.

Keywords

Generative AI, LLM, software engineering, productivity, code assistant

*Work conducted while an employee of IBM Software, Kochi, India.

CHI EA '25, Yokohama, Japan

ACM ISBN 979-8-4007-1395-8/25/04
https://doi.org/10.1145/3706599.3706670

ACM Reference Format:
Justin D. Weisz, Shraddha Vijay Kumar, Michael Muller, Karen-Ellen Browne, Arielle Goldberg, Katrin Ellice Heintze, and Shagun Bajpai. 2025. Examining the Use and Impact of an AI Code Assistant on Developer Productivity and Experience in the Enterprise. In *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems (CHI EA '25), April 26–May 01, 2025, Yokohama, Japan.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3706599.3706670

1 Introduction

AI assistants powered by large language models (LLMs) are becoming increasingly prevalent in the workplace. A number of commercial and open-source coding assistants have been released for software engineers, developers, and data scientists[1] who perform code-related work. These tools include GitHub Copilot[2], Amazon Q Developer[3], Gemini Code Assist[4], and IBM's watsonx Code Assistant[5].

With this rapid proliferation of AI code assistants, it is important to understand their impact on developer productivity. We were provided with the opportunity to examine a new AI assistant under development within IBM in 2024 that supported general developer needs in common programming languages (e.g. Python, Java, JavaScript, C++, and more). Our team – spanning product management, design, and research – used a mixed-methods approach to characterize the assistant's impact on productivity.

In this paper, we report selected results from two studies: a large-scale survey of WCA users and small-scale usability testing. We discovered several interesting insights regarding the use of the assistant, the content it generated, and its impact on productivity and the developer profession. Our paper makes the following contributions to the CHI community:

[1]In this paper, we use the term "developer" as a catch-all to cover individuals who perform code-related work, including software engineers, architects, and data scientists.
[2]GitHub Copilot: https://github.com/features/copilot
[3]Amazon Q Developer: https://aws.amazon.com/q/developer/
[4]Gemini Code Assist: https://cloud.google.com/gemini/docs/codeassist/overview
[5]watsonx Code Assistant: https://www.ibm.com/products/watsonx-code-assistant

- We characterize the user experience of an LLM-based programming assistant under development within a large technology company. Our work examines the impact of the assistant on attitudinal measures of perceptions of productivity, complementing prior work that examined behavioral productivity metrics (e.g. [61, 62]). We find that although the assistant did increase net productivity despite variability in the quality of its outputs, those gains were not evenly distributed across all users.
- We observed that *understanding code* was the top use case, followed by the *production of code*, indicating a need for further research in how AI code assistants can aid sensemaking tasks in code repositories.
- We identify a shared responsibility between people and AI systems in mitigating the risks of generated outputs.

2 Related Work

We outline three areas relevant to our study of AI code assistants: code-fluent LLMs and their incorporation into the software engineering workflow; the multi-faceted nature of productivity in software engineering; and studies of AI code assistants.

2.1 Code-fluent LLMs and software engineering assistants

Large language models that have been exposed to source code in their pre-training have demonstrated a high degree of aptitude in performing a variety of tasks: converting natural language to code (e.g. [2, 16, 19, 56]), converting code to natural language documentation (e.g. [16, 31]) or explanations (e.g. [37]), and converting code to code, such as by translating it from one language to another (e.g. [2, 19, 46, 57]) or by creating unit tests (e.g. [48]). The introduction of the Codex model [10] and its corresponding incorporation into software developers' IDEs through GitHub Copilot[6] demonstrated how code-fluent LLMs could revolutionize the software development workflow. New agentic design patterns are enabling coding assistants to perform even more complex tasks, such as converting issues into pull requests [25] and new feature descriptions into specifications and implementation plans[7].

2.2 Software engineering productivity

Productivity in software engineering is a complex, multi-faceted construct [17, 47]. It is often assessed via objective metrics of productivity that capture the ratio of output to effort [47] (e.g. lines of code over time [15], function points [30]), the complexity of the software system [20, 33], or the presence of errors or defects [26]. However, Meyer et al. [34] considers "when software developers perceive themselves to be productive and... unproductive" [34, p.1] as an important aspect of productivity.

Cheng et al. [11] outline a number of subjective and objective factors that impact developer productivity, including code quality, technical debt, infrastructure tools and support, team communication, and organizational changes and processes. In addition, researchers have found correlations between subjective and objective productivity metrics, such as the acceptance rate of suggested code [62] and the number of source code files owned by a developer [40] being correlated with perceived productivity.

The comprehensive landscape of software engineering productivity is captured by the SPACE framework [17], which outlines both objective and subjective metrics across individuals, teams, and organizations. In this paper, we focus on attitudinal and human-centered measures of productivity such as self-efficacy [44] and the impact of AI on the work process [53].

2.3 Impact of LLM-based assistants on developer productivity

Many studies have been conducted to examine the impact of LLM-based coding assistants on various aspects of productivity, albeit with mixed results [13, 23, 27, 38, 41, 44, 50, 53, 54, 58, 60–62]. One early study by Weisz et al. [53] examined AI-assisted code translation and found a net benefit to working with AI, though that benefit was not equally experienced by all participants. Kuttal et al. [27] examined human-human and human-AI pair-programming teams but did not find strong differences in outcomes such as productivity, code quality, or self-efficacy.

Ziegler et al. examined the impact of GitHub Copilot on developer productivity [61, 62] and found that developers who used the tool self-reported higher levels of productivity. Contrarily, studies by Imai [23] and GitClear [18] both suggest that the quality of the code produced by GitHub Copilot may be harming productivity due to the number of lines that must be changed or deleted.

Another consideration for AI code assistants is their impact on the work process. Both Barke et al. [5] and Liang et al. [28] identified two complementary types of usage of GitHub Copilot: "acceleration mode" in which the tool aided developers when they knew what to do next, and "exploration mode" to help developers brainstorm potential solutions to coding problems when they were unsure of how to proceed.

3 Case Study of an AI Code Assistant

IBM's watsonx Code Assistant (WCA) is family of software engineering assistants that supports enterprise-specific use cases including IT automation[8] and mainframe application modernization[9]. In mid-2024, a new variant of WCA, known as "WCA@IBM," was released internally within IBM and was rapidly adopted by over 12,000 IBM developers. This variant provided general programming assistance in languages including Python, Java, JavaScript, C++, and more. It was implemented as plugins to VSCode and Eclipse, and it supported code generation from natural language, code autocompletion, code explanation and documentation, unit test generation, and conversational Q&A.

We had the opportunity to study internal usage of this new WCA[10] variant from a subjective standpoint: how did it impact developers' perceptions of their own productivity? To address this

[6] GitHub Copilot: https://github.com/features/copilot
[7] GitHub Copilot Workspace: https://githubnext.com/projects/copilot-workspace

[8] IBM watsonx™ Code Assistant for Red Hat® Ansible® Lightspeed: https://www.ibm.com/products/watsonx-code-assistant-ansible-lightspeed
[9] IBM watsonx™ Code Assistant for Z: https://www.ibm.com/products/watsonx-code-assistant-z
[10] For simplicity, we refer to the internal deployment of "WCA@IBM" within IBM as "WCA" in this paper and we note that our use of "WCA" is not intended to refer to other products in the watsonx Code Assistant family.

question, we used a two-pronged approach: we conducted a survey of internal WCA users (N=669 respondents) and we conducted unmoderated usability testing in which participants used multiple WCA features to complete small programming tasks (N=15 participants).

Due to privacy concerns and sensitivities in data collection, we were only able to analyze attitudinal data in our studies. We did not have access to product telemetry or other behavioral data regarding use of WCA, such as the number of queries made throughout the study period, prompts sent to the model, or outputs generated by the model.

3.1 Survey

We developed a comprehensive survey to capture a wide range of data regarding the user experience of WCA and its impact on developer productivity. It was developed to meet the needs of multiple stakeholder groups, including product managers, designers, and researchers. We outline the different categories of measures in Table 1. We used a number of measures to capture different aspects of the WCA user experience, which we detail in Appendix A.

Given the large number of measures included in the survey, we split it into two modules to prioritize the most important questions and help respondents avoid survey fatigue. Module 1 focused on understanding usage, usability, and demographics, and Module 2 focused more in-depth on motivations for use, trust, and content ownership & responsibility. Each module took approximately 15-20 minutes to complete. Between Modules 1 and 2, respondents were asked whether they wanted to end the survey or spend additional time to provide more in-depth feedback.

3.1.1 Recruitment & participants. We launched a first round of data collection (Cohort 1) in May 2024 and received 105 responses. We launched a second round of data collection (Cohort 2) in July 2024, targeted to a group of 564 developers participating in a WCA training program. As completing the survey was part of the program, we eliminated the optional nature of Module 2 for this cohort.

A total of 669 WCA users responded to our survey. Despite the optional nature of Module 2 for Cohort 1, only 41 respondents did not complete this module (39.0% of Cohort 1; 6.1% of the total sample). Because we do not hypothesize any differences between the cohorts, we analyze them as a single group[11].

Survey respondents held a wide range of developer roles, such as back-end developers (59.6%), front-end developers (19.9%), QA/test developers (19.3%), and more. They held a wide range of tenures with IBM (< 6 months to 31+ years), years of experience working as a professional software engineer (0 to 10+ years), and hailed from a variety of geographies (primarily Americas and APAC). We show distributions of respondent demographics in Appendix B.

3.2 Unmoderated usability testing

We conducted an unmoderated usability test to assess specific features of WCA. This test involved solving a small programming problem using different WCA features: code generation, chat, and code autocompletion. Participants then used WCA to generate explanations for their code and documentation in a README file. After each task, participants filled out a small questionnaire that asked about their experience. The usability test took approximately 40 minutes.

3.2.1 Recruitment & participants. We recruited 15 WCA users to complete the usability test based on the programming languages with which they worked and the frequency of their WCA usage. Participants came from a variety of product teams, spanning areas including firmware development, security, mobile apps, databases, and more. About 43% of participants used WCA regularly in their work.

3.3 Ethics statement

Our research followed IBM's AI Ethics framework[12]. Our survey was approved by an internal review committee, subject to restrictions regarding the collection of demographic information. Specifically, we were unable to collect age or gender identity, we collected geography and job role using specified sets of options, and we did not collect any personally-identifiable information (PII). All survey responses were anonymous[13].

4 Results

Our data consists of a mix of quantitative measures from the survey and qualitative feedback from the survey and the unmoderated usability testing sessions. We generally report descriptive statistics on the quantitative measures as our research questions did not require making statistical comparisons. To analyze qualitative data, we conducted a reflexive thematic analysis [12], using a deductive approach as our product management and design stakeholders had clear priorities for which aspects of the user experience on which to focus. We summarize our key results in Table 1.

In presenting our results, we refer to survey respondents as `Rx.yyy`, where `x` corresponds to their cohort (1 or 2) and `yyy` is a unique identifier. We refer to participants in the unmoderated usability study as `Pxx`, where `xx` is a unique identifier. Collectively, we refer to both groups as "WCA users."

4.1 Motivations, use, and non-use

Liang et al. [28] examined a number of motivations for developers to use (or not use) AI programming assistants. Compared to their sample of AI code assistant users *in the wild*, our respondents were less interested in having a code autocompletion feature (47% vs. 86%) or finishing their programming tasks faster (59% vs. 76%). Rather, they were more interested in discovering potential ways or starting points to solve their programming challenges (63% vs. 50%). In addition, our respondents were interested in exploring new tools (68%), they wanted to know how their work might change in the future (64%), they felt a responsibility to try new IBM products (64%),

[11]We note that, despite the gap in time between the two cohorts, no major product enhancements beyond minor bug fixes were deployed; thus, the user experience between the two cohorts was equivalent.

[12]IBM AI Ethics: https://www.ibm.com/impact/ai-ethics

[13]For Cohort 2 respondents, we preserved the anonymity of their feedback by collecting their email address in a separate form after they completed the survey to mark their completion of the training course. In addition, it is possible that an individual responded to both surveys. However, given that the number of respondents in Cohort 2 was vastly greater than that of Cohort 1, we were not overly concerned with the possibility of capturing repeated measures from the same set of respondents.

Category	Description	Summary of Findings	Section
Motivations, use, and non-use	Why and how WCA was used or not used	Top use cases focused on code understanding; "off-label" usage by content designers; unmet needs for specialized technologies (e.g. DB2, Maximo)	4.1
Use of generated content	Ways that generated content was reviewed & used	Content modified before use; outputs also used for learning and inspiration	4.2
Impact on productivity	Impact of WCA on various dimensions (effort, speed, work quality, self-efficacy)	Small net productivity improvement, but with mixed and disparate impact	4.3
Authorship & responsibility	Who deserves authorship credit and who is responsible for avoiding inclusion of copyrighted IP?	WCA deserving of authorship credit for co-creative activity; users and WCA have a joint responsibility to avoid inclusion of copyrighted IP	4.4
Impact on job role	How AI assistants might change the developer profession	AI lets developers focus on higher-level tasks; potential for deskilling; increased productivity translates into increased expectations	4.5

Table 1: Categories of measures included in the user experience survey and a summary of our findings. We provide a complete listing of all survey questions in Appendix A.

and their management recommended using WCA (60%). We provide a more detailed comparison of motivations for use in Appendix C.

We asked respondents about the kinds of tasks they conducted with WCA within their prior two weeks of usage. Prior research on LLM-based code assistants suggests that the core use is to *generate code* [28]. By contrast, the most popular use cases for WCA related to *understanding* code, either via explanations (71.9%) or by receiving answers to general programming questions (68.5%). R1.92 summarized the utility of WCA for code understanding: *"I use WCA for two main things: Explaining code other people wrote that I am working with for the first time. Explaining code I have written that isn't working as expected, because sometimes WCA can tell me why it isn't working and give tips on how to fix it. It can often catch typo-like errors that I overlooked."* R1.45 pointed out how explanations saved them time: *"I like its ability to explain functions of code which could take a bit to understand. It can save a lot of time."* R2.177 described their use of explanations to *"explore areas of code I'm not familiar with."* Respondents did use WCA to generate code (55.6%), documentation (39.6%), and tests (35.7%), but to a lesser extent. We detail additional purposes of use in Appendix D.

Some respondents used WCA to improve specific qualities or characteristics of code, including its readability (17.9%), maintainability (15.2%), performance (12.8%), and security (6.7%). Usability test participants also indicated the importance of qualities including readability, maintainability, reliability, scalability, and testability. These desires for AI assistance beyond mere code generation mirror user needs for controlling code-fluent LLM outputs discovered by Houde et al. [21].

A small number of respondents reported not making use of WCA within the prior two weeks (N=28; 4.2%). They indicated that it was faster to do the work themselves (39.3%), they didn't feel that WCA provided helpful suggestions (32.1%), they hadn't conducted any code-related work within the prior two weeks (25.0%), and WCA didn't support their functional or non-functional requirements (14.3%). Multiple respondents indicated a need for WCA to handle the specialized technologies with which they worked, such as R2.279, who commented, *"I am a db2 developer and WCA's knowledge on db2 internals is poor."* R2.365 also noted, *"it does not seem a good tool to search about IBM Products/offerings... such as Maximo Application Suite and MAS Ansible DevOps automation."* In enterprise scenarios, it may be even more important to support the specialized technologies used in these environments to increase the utility of such assistants.

One interesting theme that stood out to us was a reluctance to use WCA because it may reflect negatively upon the user. R1.49 explained, *"I just don't use code generated by WCA@IBM... obviously I would not want to be seen with generated code in my PRs, so embarrassing!"* This sentiment was echoed by R1.92, who said, *"I imagine some people could find it embarrassing because the tool is so new and very few people on my team use it. There is an inherent suspicion against AI-generated code."* Although only two respondents raised this issue, it may be important for organizational leaders to foster a culture in which AI assistance is viewed as acceptable to garner wide-spread adoption.

Finally, we discovered a small group of technical writers who found utility in using WCA to understand technical concepts without having to disturb their developer counterparts. R1.57 remarked, *"My favorite feature is to understand the technical terms and code provided by the Dev. This reduces the time in understanding the API terms and code rather than discussing with the Dev."* Multiple respondents desired using WCA to *"create customer facing documentation"* (R1.27) and *"help me writ[e] drafts following IBM content guidelines."* (R1.68). These "off-label" use cases by people in developer-adjacent roles surprised us and indicate the importance of taking a holistic view on the potential beneficiaries of AI code assistants.

4.2 Use of generated content

Respondents reported using generated outputs – code, documentation, unit tests, and explanations – in different ways. Overall,

the use of generated outputs without modification was not common (2-4% of respondents reported doing this, depending on output type); rather, respondents often modified outputs before using them (9-19%) or used them for another purpose, such as learning something new (23-35%) or getting new ideas (24-37%). Users described how *"the results give me new ideas"* (R2.180) and *"It is very helpful to get started writing code in a new language"* (R2.626) by *"recommend[ing] an approach I haven't thought of or I wasn't even aware of"* (R2.405). P1 described how *"creating diagrams in markdown works from the code."* Users also talked about how WCA helped them recall *"concepts which may be I have forgotten during [the] course of time"* (R2.296) and aiding them when *"[I] know what to do, but don't know how to do it or forgot about that."* (R2.292). These uses reinforce the value that generative AI provides in helping people learn [1, 6, 49, 59].

4.3 Impact on productivity

We evaluated the impact of WCA on respondents' perceptions of productivity using multiple measures: 7-point semantic differential scales[14] that assessed effort, quality, and speed [53] and a 4-item scale of self-efficacy[15] (derived from [44]). Overall, respondents felt that WCA made their work easier (M (SD) = .78 (1.45), $t(609) = 13.35$, $p < .001$, 95% CI = [.67, .90]), of a better quality (M (SD) = .66 (1.25), $t(603) = 13.02$, $p < .001$, 95% CI = [.56, .76]), and faster (M (SD) = .57 (1.48), $t(606) = 9.57$, $p < .001$, 95% CI = [.46, .69]), as indicated by means and 95% CIs > 0. However, the magnitudes of productivity improvements were small, further evidenced by self-efficacy ratings falling around the scale midpoint (M (SD) = 3.20 (.93) of 5).

Despite net positive ratings, the benefits of WCA were not evenly distributed: 42.6% of respondents felt that WCA made them less effective (self-efficacy ≤ 3), whereas 57.4% of respondents felt that WCA made them more effective (self-efficacy > 3). Users' comments shed light on WCA's mixed impact on productivity. Some users benefited from using WCA in *acceleration mode*: *"Its ability to suggest code and code autocompletion... improves my work productivity significantly."* (R2.608). P11 remarked, *"It saves time compared to write from scratch by myself."* WCA helped R2.236 come up to speed faster in a new project: *"I used WCA to help document and explain me new functionality in different classes to help understand it quicker, thus making my productivity faster."* Other users were helped in *exploration mode* when WCA *"provid[es] different approaches towards the problem"* (P11), which R2.311 described as *"my favourite feature."* P6 also felt, *"I like the code generation aspect... it helps me think about the solution."*

Contrarily, imperfections in the quality of WCA's output may require additional user effort to identify and correct. R1.25 pointed out that, *"Sometimes WCA goes off topic and it ends up wasting time that could've been used to finish up the work. If multiple retries are needed to get the desired result, it becomes counter-productive."* (R1.25). P5 noted that *"It hallucinated"* during the usability test. R2.658 highlighted the relationship between correctness and trust, saying, *"If it doesn't have close to 100% correctness, then I cannot trust its answers on topics that I don't actually know myself. It is limited to helping me speed up more routine tasks that I can do myself."* R2.603 commented, *"You have to spend time to verify it."* P13 similarly felt, *"WCA created code need[s] to be tested,"* and R2.183 lamented, *"it is a burden to have to double check answers...I still don't have enough confidence to blindly trust the responses."*

We assessed the quality of WCA's outputs on a 5-point scale[16]: Very poor (1), Poor, Acceptable, Good, Very good (5). Respondents rated WCA's quality in the middle of the scale (M (SD) = 3.20 (.77)), indicating an "acceptable" quality but with room for improvement. This level of quality was good enough for some respondents, but not others:

> *"I'm really impressed with the quality of simple python programs that can be generated."* (R2.143)
>
> *"The code generated is of poor quality and often does not meet the stated requirements."* (R1.62)

Another factor that impacted perceptions of productivity was the speed of WCA's responses. We assessed speed on a 5-point scale: Very slow (1), Slow, Acceptable, Fast, Very fast (5). Speed was rated slightly below the scale midpoint (M (SD) = 2.88 (0.86)) and many respondents commented on how it needed to be improved: *"the code suggestion needs to be faster. At the moment it does not keep up with my typing."* (R1.61).

It is clear that WCA had an impact on productivity, but its impact was mixed and disparate. Several respondents pointed out how they viewed the role of WCA as an "intern" or "junior developer" given its current performance:

> *"In some ways I feel it's like an intern that just started on the project and does training on the job. Needs a lot of supervision, but can be handed some set of tasks."* (R1.86)
>
> *"I tend to view it as a junior developer helping me out. It can generate code much more quickly than a junior developer, and usually of higher quality, but there are still often mistakes, edge cases, etc. that need to be fixed. So I always need to be mindful that the generated code has to be carefully reviewed and often manually updated before it can be used."* (R1.11)

4.4 Authorship & responsibility

Working with WCA is a co-creative process [36, 43, 55] in which both human users and WCA are capable of shaping an artifact-under-production, source code. Given the novelty of this form of interaction, we were interested in understanding how WCA impacted developers' perceptions of their authorship[17] over code co-produced with WCA.

We considered how respondents felt about who authored code in four different scenarios: (1) reviewing WCA-generated code but

[14] These scales were coded from [-3, +3] such that ratings > 0 correspond to beneficial impact, ratings of 0 indicate no impact, and ratings < 0 correspond to detrimental impact.

[15] Items were rated on 5-point Likert scales. A confirmatory factor analysis indicated this scale was highly reliable (Cronbach's $\alpha = .91$).

[16] Quality was assessed separately for each type of generated content: code, documentation, unit tests, explanations, and Q&A responses. These items formed a scale with high reliability (Cronbach's $\alpha = 0.83$ and thus were averaged into a single metric of quality.

[17] Authorship and ownership are related, but distinct concepts. Ownership provides certain rights over the work, such as holding the copyright; as our users are employed by a corporation that owns the intellectual property produced by their employees, we assessed feelings of authorship rather than ownership.

implementing the functionality themselves; (2) pasting in WCA-generated code verbatim; (3) pasting in WCA-generated code but then modifying it; and (4) implementing an idea suggested by WCA.

Many respondents viewed themselves as the sole author when implementing functionality themselves (scenario 1: 57.5%). Similarly, many respondents viewed WCA as the sole author when they pasted its outputs verbatim (scenario 2: 53.7%). Interestingly, some respondents felt a *shared sense of authorship* across all scenarios: when it produced code that they rewrote (scenario 1: 30.4%), when they used its outputs verbatim (scenario 2: 27.8%), when each party contributed to the code (scenario 3: 64.4%), and even when WCA only contributed ideas (scenario 4: 39.8%). These results suggest that new mechanisms may be needed to track co-creative activities and ensure that each party's contributions – human and AI – are properly attributed in ways that do not *"steal the recognition and hardwork of the developers"* (R2.632).

Another aspect of code authorship concerns the responsibility for mitigating the risks of generated outputs [22], such as ensuring generated code does not violate the IP rights of other parties. Given the current legal climate concerning the reproduction of copyrighted works by LLMs (e.g. [9, 45]), we also wanted to understand the extent to which our users felt responsible for ensuring that WCA did not reproduce copyrighted works[18].

Liang et al. [28] found that 46% of their participants had concerns that AI programming assistants may produce code that infringes on intellectual property. By contrast, most of our respondents (83.4%) expressed similar concerns over WCA reproducing copyrighted materials not owned by IBM. In addition, most respondents felt that they had a responsibility (89.2%) and that WCA had a responsibility (96.2%) to ensure copyrighted materials were not included in their source code. R2.549 aptly summarized their own responsibilities: *"...if I am the person who uses the tool, it's my responsibility know what is going to be part of the code or not."* Conversely, R2.459 expected WCA to detect copyrighted material: *"it seems like copyrighted material should be detected as copyrighted and if this is not something WCA can do, it should definitely immediately do [so]."* Many respondents expected that, *"all content generated or reproduced by WCA@IBM adheres to copyright regulations,"* (R2.652) and *"WCA needs to fully own responsibility for not reproducing copyrighted code"* (R1.86). Thus, protecting the integrity of a codebase is seen as a shared responsibility, and one for which developers may require new kinds of support.

4.5 Impact on job role

Respondents held a wide range of views on how WCA would change their job role, responsibilities, and purpose as a developer. Some users felt that there would be "No Change" (R1.13), or that *"It does not [change my role]. Yet, anyway :o)"* (R1.52), because *"I'm still better than WCA"* (R2.546) or *"I see it as a tool to better my work experience"* (R2.142). Other users felt that their role would change, but weren't sure of how: *"It confuses me on what my duty is as the boundary is not clear"* (R2.472). Echoing sentiments of WCA's role as a "junior developer" or "intern," some users felt it would free them up to focus on the higher-level, creative aspects of their profession:

> *"I believe WCA has large potential to generate boilerplate code and implement trivial pieces of code. This would save developers time and effort writing a lot of repetitive and mundane code. It will allow developers to focus on solving the harder, higher level, and more creative problems."* (R1.24)

R2.304 also pointed out that having an AI assistant focus on the *"many dirty repeated tasks like security fixes, add unit test cases/ translate the unit test cases etc."* would allow them to *"save my time to focus on innovative feature design and development."*

Users were sensitive to the potential deskilling aspects of AI assistance [42, 51]. R2.185 espoused a view that, *"[I] [d]on't really like to use AI, makes people lazy and promotes to concept of not to think."* R2.547 expressed the same concern, albeit more bluntly: *"I suspect we're all going to get a lot stupider, doing a worse job of maintaining larger amounts of worse code."* However, technological acceptance takes time [14], and new technologies have a learning curve before people can use them effectively; for LLMs, effective use requires learning the art of prompt engineering [29, 32]. WCA is no exception: *"Very few people on my team have tried it, and many aren't sure how to engineer prompts to get effective answers yet"* (R1.92).

One final impact we noticed on the role of the developer is a consequence of AI augmenting human abilities: when a developer is more productive with AI, the expectations of their productivity may also be increased. This sentiment was captured by P6, who said, *"Since our team is expected to use WCA our management is expecting more work in one sprint as compared to before."*

5 Discussion & Conclusions

Our examination of WCA usage revealed a number of insights on the use of AI code assistants within the enterprise and their impact on developer productivity.

- **Code understanding is a top use case.** We anticipated that code generation would be a top use case due to the nature of generative AI and previous examinations of AI code assistants [28]. Rather, WCA was primarily used to support code understanding, either via explanations of existing code or through answers to general questions. This observation motivates the need for further research into how AI code assistants can help developers conduct sensemaking tasks in existing code repositories and bolster their knowledge of programming languages and concepts.
- **Software engineering is a co-creative activity.** Users generally did not accept the code produced by WCA without first reviewing and modifying it, suggesting that overreliance problems may not be as prevalent with AI code assistants as in other domains [3, 4, 8]. Additionally, they felt a shared sense of authorship over the code produced with WCA, even in instances when WCA only provided ideas and not actual source code. This observation suggests that new mechanisms for tracking authorship attribution may be necessary to give

[18] Our examination of this topic is not predicated on observations of WCA actually reproducing copyrighted works; Mishra et al. [35] provide an explanation of the data used to train the underlying Granite model.

each party proper credit for their work, in addition to tracking the provenance of contributions within a larger source code repository.

- **Perfection is not required.** Echoing the title of the paper by Weisz et al. [52], we similarly observed that (some) developers felt productivity improvements from WCA despite variability in its quality and speed. We anticipate that quality and speed improvements will help more developers receive these benefits. We also observed that some developers may require additional training in effectively prompting LLMs.
- **Mitigating risks in generated outputs is a joint responsibility.** LLMs have the potential to reproduce materials contained within their training data [7]; in the domain of software engineering, there is a risk of contaminating a codebase with copyrighted material or code subject to specific licenses (e.g. GPL). Users felt that they themselves, as well as WCA, were responsible for minimizing this risk. New socio-technical approaches that combine algorithmic methods (e.g. code similarity detection [24, 39]) with intelligent, human-driven review interfaces, may be needed to minimize these risks.
- **Developers are worried about deskilling and early adoption.** Users were concerned that their use of AI assistants may result in a loss of skills. In addition, two users were reluctant to use WCA due to perceived negative social consequences of being an early adopter. These findings indicate a need to more clearly articulate the benefits of AI code assistants and an organization's stance on their use: the developer profession will change to focus on higher-level, creative aspects of the work, and clear organizational policies can help employees feel more comfortable using AI code assistants in their work.

Our case study represents the use of a specific AI code assistant at a specific point of time (the summer of 2024). With the rapid pace of advancement in AI, we anticipate some issues experienced by our users, especially regarding quality and speed, will naturally diminish as model performance increases. What won't change is the need for human ingenuity and insight to determine *what* software systems to build, even if the mechanics of *how* those systems are built are increasingly aided by AI.

Acknowledgments

We thank Katharina Schippert and Robin Auer for their support in defining the WCA user research program and recruiting participants for our studies. We also thank Keri Olson and Melissa Modjeski whose support made this research possible. Finally, we thank all of our users who provided us with valuable feedback.

References

[1] Ibrahim Adeshola and Adeola Praise Adepoju. 2023. The opportunities and challenges of ChatGPT in education. *Interactive Learning Environments* (2023), 1–14.

[2] Wasi Uddin Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2021. Unified pre-training for program understanding and generation. *arXiv preprint arXiv:2103.06333* (2021).

[3] Zahra Ashktorab, Michael Desmond, Josh Andres, Michael Muller, Narendra Nath Joshi, Michelle Brachman, Aabhas Sharma, Kristina Brimijoin, Qian Pan, Christine T Wolf, et al. 2021. Ai-assisted human labeling: Batching for efficiency without overreliance. *Proceedings of the ACM on Human-Computer Interaction* 5, CSCW1 (2021), 1–27.

[4] Zahra Ashktorab, Qian Pan, Werner Geyer, Michael Desmond, Marina Danilevsky, James M Johnson, Casey Dugan, and Michelle Bachman. 2024. Emerging Reliance Behaviors in Human-AI Text Generation: Hallucinations, Data Quality Assessment, and Cognitive Forcing Functions. *arXiv preprint arXiv:2409.08937* (2024).

[5] Shraddha Barke, Michael B James, and Nadia Polikarpova. 2023. Grounded copilot: How programmers interact with code-generating models. *Proceedings of the ACM on Programming Languages* 7, OOPSLA1 (2023), 85–111.

[6] Brett A Becker, Michelle Craig, Paul Denny, Hieke Keuning, Natalie Kiesler, Juho Leinonen, Andrew Luxton-Reilly, James Prather, and Keith Quille. 2023. Generative ai in introductory programming. *Name of Journal* (2023).

[7] Emily M Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. 2021. On the dangers of stochastic parrots: Can language models be too big?. In *Proceedings of the 2021 ACM conference on fairness, accountability, and transparency*. 610–623.

[8] Zana Buçinca, Maja Barbara Malaya, and Krzysztof Z Gajos. 2021. To trust or to think: cognitive forcing functions can reduce overreliance on AI in AI-assisted decision-making. *Proceedings of the ACM on Human-computer Interaction* 5, CSCW1 (2021), 1–21.

[9] Matthew Butterick. 2022. *GitHub Copilot Litigation.* Retrieved 04-Oct-2024 from https://githubcopilotlitigation.com

[10] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374* (2021).

[11] Lan Cheng, Emerson Murphy-Hill, Mark Canning, Ciera Jaspan, Collin Green, Andrea Knight, Nan Zhang, and Elizabeth Kammer. 2022. What improves developer productivity at google? code quality. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1302–1313.

[12] Victoria Clarke and Virginia Braun. 2017. Thematic analysis. *The journal of positive psychology* 12, 3 (2017), 297–298.

[13] Arghavan Moradi Dakhel, Vahid Majdinasab, Amin Nikanjam, Foutse Khomh, Michel C Desmarais, and Zhen Ming Jack Jiang. 2023. Github copilot ai pair programmer: Asset or liability? *Journal of Systems and Software* 203 (2023), 111734.

[14] Fred D Davis, RP Bagozzi, and PR Warshaw. 1989. Technology acceptance model. *J Manag Sci* 35, 8 (1989), 982–1003.

[15] Prem Devanbu, Sakke Karstu, Walcelio Melo, and William Thomas. 1996. Analytical and empirical evaluation of software reuse metrics. In *Proceedings of IEEE 18th International Conference on Software Engineering*. IEEE, 189–199.

[16] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. 2020. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155* (2020).

[17] Nicole Forsgren, Margaret-Anne Storey, Chandra Maddila, Thomas Zimmermann, Brian Houck, and Jenna Butler. 2021. The SPACE of Developer Productivity: There's more to it than you think. *Queue* 19, 1 (2021), 20–48.

[18] GitClear. [n. d.]. *Coding on Copilot: 2023 Data Suggests Downward Pressure on Code Quality.* Retrieved 24-Sep-2024 from https://www.gitclear.com/coding_on_copilot_data_shows_ais_downward_pressure_on_code_quality

[19] Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie Liu, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, et al. 2020. Graphcodebert: Pre-training code representations with data flow. *arXiv preprint arXiv:2009.08366* (2020).

[20] Maurice H Halstead. 1977. *Elements of Software Science (Operating and programming systems series).* Elsevier Science Inc.

[21] Stephanie Houde, Vignesh Radhakrishna, Praneeth Reddy, Juie Darwade, Haoran Hu, Kalpesh Krishna, Mayank Agarwal, Kartik Talamadupula, and Justin Weisz. 2022. User and technical perspectives of controllable code generation. In *Annual Conference on Neural Information Processing Systems*.

[22] IBM. 2024. *AI Risk Atlas.* Retrieved 07-Oct-2024 from https://www.ibm.com/docs/en/watsonx/saas?topic=ai-risk-atlas

[23] 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*. 319–321.

[24] Sarthak Jain, Aditya Dora, Ka Seng Sam, and Prabhat Singh. 2024. LLM Agents Improve Semantic Code Search. *arXiv preprint arXiv:2408.11058* (2024).

[25] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2023. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770* (2023).

[26] Amy J Ko and Brad A Myers. 2005. A framework and methodology for studying the causes of software errors in programming systems. *Journal of Visual Languages & Computing* 16, 1-2 (2005), 41–84.

[27] Sandeep Kaur Kuttal, Bali Ong, Kate Kwasny, and Peter Robe. 2021. Trade-offs for substituting a human with an agent in a pair programming context: the good,

the bad, and the ugly. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–20.

[28] 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 46th IEEE/ACM International Conference on Software Engineering*. 1–13.

[29] Leo S Lo. 2023. The art and science of prompt engineering: a new literacy in the information age. *Internet Reference Services Quarterly* 27, 4 (2023), 203–210.

[30] Graham C. Low and D. Ross Jeffery. 1990. Function points in the estimation and evaluation of the software process. *IEEE transactions on Software Engineering* 16, 1 (1990), 64–71.

[31] Qinyu Luo, Yining Ye, Shihao Liang, Zhong Zhang, Yujia Qin, Yaxi Lu, Yesai Wu, Xin Cong, Yankai Lin, Yingli Zhang, et al. 2024. Repoagent: An llm-powered open-source framework for repository-level code documentation generation. *arXiv preprint arXiv:2402.16667* (2024).

[32] Ggaliwango Marvin, Nakayiza Hellen, Daudi Jjingo, and Joyce Nakatumba-Nabende. 2023. Prompt engineering in large language models. In *International conference on data intelligence and cognitive informatics*. Springer, 387–402.

[33] Thomas J McCabe. 1976. A complexity measure. *IEEE Transactions on software Engineering* 4 (1976), 308–320.

[34] André N Meyer, Thomas Fritz, Gail C Murphy, and Thomas Zimmermann. 2014. Software developers' perceptions of productivity. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. 19–29.

[35] Mayank Mishra, Matt Stallone, Gaoyuan Zhang, Yikang Shen, Aditya Prasad, Adriana Meza Soria, Michele Merler, Parameswaran Selvam, Saptha Surendran, Shivdeep Singh, et al. 2024. Granite code models: A family of open foundation models for code intelligence. *arXiv preprint arXiv:2405.04324* (2024).

[36] Caterina Moruzzi and Solange Margarido. 2024. A user-centered framework for human-ai co-creativity. In *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*. 1–9.

[37] 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*. 1–13.

[38] 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*. 1–5.

[39] Matija Novak, Mike Joy, and Dragutin Kermek. 2019. Source-code similarity detection and detection tools used in academia: a systematic review. *ACM Transactions on Computing Education (TOCE)* 19, 3 (2019), 1–37.

[40] Edson Oliveira, Eduardo Fernandes, Igor Steinmacher, Marco Cristo, Tayana Conte, and Alessandro Garcia. 2020. Code and commit metrics of developer productivity: a study on team leaders perceptions. *Empirical Software Engineering* 25 (2020), 2519–2549.

[41] Ruchika Pandey, Prabhat Singh, Raymond Wei, and Shaila Shankar. 2024. Transforming Software Development: Evaluating the Efficiency and Challenges of GitHub Copilot in Real-World Projects. *arXiv preprint arXiv:2406.17910* (2024).

[42] Janet Rafner, Dominik Dellermann, Arthur Hjorth, Dóra Verasztó, Constance Kampf, Wendy Mackay, and Jacob Sherson. 2022. Deskilling, upskilling, and reskilling: a case for hybrid intelligence. *Morals & Machines* 1, 2 (2022), 24–39.

[43] Jeba Rezwana and Mary Lou Maher. 2023. User perspectives on ethical challenges in Human-AI co-creativity: A design fiction study. In *Proceedings of the 15th Conference on Creativity and Cognition*. 62–74.

[44] 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*. 491–514.

[45] Emma Roth. 2024. The developers suing over GitHub Copilot got dealt a major blow in court. *The Verge* (9 July 2024). Retrieved 04-Oct-2024 from https://www.theverge.com/2024/7/9/24195233/github-ai-copyright-coding-lawsuit-microsoft-openai

[46] Baptiste Roziere, Marie-Anne Lachaux, Lowik Chanussot, and Guillaume Lample. 2020. Unsupervised translation of programming languages. *Advances in neural information processing systems* 33 (2020), 20601–20611.

[47] Caitlin Sadowski and Thomas Zimmermann. 2019. *Rethinking productivity in software engineering*. Springer Nature.

[48] Max Schäfer, Sarah Nadi, Aryaz Eghbali, and Frank Tip. 2023. An empirical evaluation of using large language models for automated unit test generation. *IEEE Transactions on Software Engineering* (2023).

[49] Shamini Shetye. 2024. An Evaluation of Khanmigo, a Generative AI Tool, as a Computer-Assisted Language Learning App. *Studies in Applied Linguistics and TESOL* 24, 1 (2024).

[50] 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 *Chi conference on human factors in computing systems extended abstracts*. 1–7.

[51] Dakuo Wang, Justin D Weisz, Michael Muller, Parikshit Ram, Werner Geyer, Casey Dugan, Yla Tausczik, Horst Samulowitz, and Alexander Gray. 2019. Human-AI collaboration in data science: Exploring data scientists' perceptions of automated AI. *Proceedings of the ACM on human-computer interaction* 3, CSCW (2019), 1–24.

[52] Justin D Weisz, Michael Muller, Stephanie Houde, John Richards, Steven I Ross, Fernando Martinez, Mayank Agarwal, and Kartik Talamadupula. 2021. Perfection not required? Human-AI partnerships in code translation. In *Proceedings of the 26th International Conference on Intelligent User Interfaces*. 402–412.

[53] 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 *Proceedings of the 27th International Conference on Intelligent User Interfaces*. 369–391.

[54] 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*. 172–178.

[55] Zhuohao Wu, Danwen Ji, Kaiwen Yu, Xianxu Zeng, Dingming Wu, and Mohammad Shidujaman. 2021. AI creativity and the human-AI co-creation model. In *Human-Computer Interaction. Theory, Methods and Tools: Thematic Area, HCI 2021, Held as Part of the 23rd HCI International Conference, HCII 2021, Virtual Event, July 24–29, 2021, Proceedings, Part I 23*. Springer, 171–190.

[56] Frank F Xu, Bogdan Vasilescu, and Graham Neubig. 2022. In-ide code generation from natural language: Promise and challenges. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 31, 2 (2022), 1–47.

[57] Zhen Yang, Fang Liu, Zhongxing Yu, Jacky Wai Keung, Jia Li, Shuo Liu, Yifan Hong, Xiaoxue Ma, Zhi Jin, and Ge Li. 2024. Exploring and unleashing the power of large language models in automated code translation. *Proceedings of the ACM on Software Engineering* 1, FSE (2024), 1585–1608.

[58] Burak Yetistiren, Isik Ozsoy, and Eray Tuzun. 2022. Assessing the quality of GitHub copilot's code generation. In *Proceedings of the 18th international conference on predictive models and data analytics in software engineering*. 62–71.

[59] Ramazan Yilmaz and Fatma Gizem Karaoglan Yilmaz. 2023. The effect of generative artificial intelligence (AI)-based tool use on students' computational thinking skills, programming self-efficacy and motivation. *Computers and Education: Artificial Intelligence* 4 (2023), 100147.

[60] Beiqi Zhang, Peng Liang, Xiyu Zhou, Aakash Ahmad, and Muhammad Waseem. 2023. Practices and challenges of using github copilot: An empirical study. *arXiv preprint arXiv:2303.08733* (2023).

[61] 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*. 21–29.

[62] Albert Ziegler, Eirini Kalliamvakou, X Alice Li, Andrew Rice, Devon Rifkin, Shawn Simister, Ganesh Sittampalam, and Edward Aftandilian. 2024. Measuring GitHub Copilot's Impact on Productivity. *Commun. ACM* 67, 3 (2024), 54–63.

A Survey instrument

In this section, we provide a listing of the survey questions discussed in this case study. Some questions included in the survey have been omitted as their relevance to this case study is low, but their inclusion was important for other internal stakeholders.

A.1 Usage within the past two weeks

The survey began by asking whether the respondent had used WCA within the past two weeks.

1. Have you used WCA within the past 2 weeks?
- Yes
- No

A.2 Motivations for use

If the respondent indicated they used WCA within the past two weeks, they were asked about their motivations for using WCA.

2a. For what purposes did you use WCA? Please check all that apply. Please respond based only on your usage of WCA within the past 2 weeks.
- Generate code (or code snippets) in the chat
- Generate code via autocompletion in the source editor
- Generate documentation
- Generate tests
- Explain code
- Answer questions about APIs or libraries
- Answer general programming questions
- Generate alternate ways to implement a method or functionality
- Refactor code
- Optimize code for performance (e.g. runtime or memory usage)
- Translate code to another programming language
- Fix code
- Improve the readability of code
- Improve the maintainability of code
- Identify security issues in code
- Leave feedback for the WCA development team

2b. For what other purposes did you use WCA? Please respond based only on your usage of WCA within the past 2 weeks.
- *Open-ended response*

A.3 Reasons for non-use

If the respondent indicated they had not used WCA in the past two weeks, they were then asked about why. These items were based on Liang et al. [28] and expanded upon by us.

3a. For what reasons have you not used WCA within the past 2 weeks? Please check all that apply.
- WCA writes code that doesn't meet functional or non-functional (e.g. security, performance) requirements that I need
- It's hard to control WCA to get code that I want
- I spent too much time debugging or modifying code written by WCA
- I don't think WCA provided helpful suggestions
- I don't want WCA to have access to my code
- I write and use code that WCA wasn't trained on which limits its ability to provide assistance
- I found WCA's suggestions too distracting
- I don't understand the code generated by WCA
- I don't understand the documentation generated by WCA
- I don't understand the explanations generated by WCA
- Code generated by WCA doesn't perform well enough for my needs
- It is faster to do the work myself
- I have not done any code-related tasks in the past 2 weeks

3b. Did you have any other reasons for not using WCA? What were they?
- *Open-ended response*

A.4 Likes & dislikes

4a. Please explain what you like about WCA. What is your favorite feature? Why?
- *Open-ended response*

4b. Please explain what you dislike about WCA@IBM. What is your least favorite feature? Why?
- *Open-ended response*

A.5 Self-efficacy

The items in this scale were derived from Ross et al. [44] and were expanded upon by us. Each item was rated on a 5-point Likert scale: Strongly disagree, Disagree, Neither disagree nor agree, Agree, Strongly agree.

5. How would you characterize your experience using WCA?
- WCA helps me write better code
- WCA helps me write code more quickly
- I feel more productive when using WCA
- I spend less time searching for information while using WCA

A.6 Speed

6. How would you characterize the speed of receiving a chat response from WCA?
- I have not used the chat feature, Unusably slow, Very slow, Slow, Acceptable, Fast, Very fast

A.7 Copyright

7a. How concerned are you that WCA may reproduce copyrighted materials not owned by IBM?
- Not concerned at all, Slightly concerned, Moderately concerned, Concerned, Very concerned

Respondents were asked to rate the degree of responsibility for each party on a 5-point scale: Not at all responsible, Slightly responsible, Moderately responsible, Responsible, Very responsible.

7b. When using WCA, to what extent are the following parties responsible for ensuring that copyrighted materials not owned by IBM are not included in IBM source code?
- Me
- WCA

A.8 Additional improvements

8. What single aspect of WCA should be immediately improved? Please explain.
- *Open-ended response*

A.9 Demographics

9a. How many years of service do you have at IBM (based on date of hire)

- < 6 months, 6 months–1 year, 1-2 years, 3-5 years, 6-10 years, 11-15 years, 16-20 years, 21-25 years, 26-30 years, 31+ years

9b. How many years have you held a role as a professional software engineer (at IBM or other companies)?

- 0-5 years, 6-10 years, 10+ years

9c. What is your job role? Please check all that apply.

- Back-End Developer
- Compiler Technology Developer
- Developer Advocate
- DevOps Developer
- Firmware Developer
- Front-End Developer
- L3 Support Engineer
- QA/Test Developer
- Software Architect
- Software Performance Analyst
- Other: *free text*

A.10 Module 2

At this point in the survey, respondents were asked whether they wished to continue to provide additional, in-depth feedback.

10. Would you like to provide additional feedback on WCA?

- Yes
- No

A.11 In-depth quality

We asked respondents to rate the quality of each type of generated output on a 5-point scale: Very poor, Poor, Acceptable, Good, Very good. These ratings were averaged to produce an overall quality score.

11. How would you characterize the quality of WCA's outputs?

- Generated code (within chat or the source editor)
- Generated unit tests
- Generated documentation
- Generated explanations of code
- Answers to general questions (within chat)

A.12 In-depth motivations for use

These motivations were derived from Liang et al. [28] and were expanded upon by us. Each item was rated on a 5-point scale: Not important at all, Slightly important, Moderately important, Important, Very important.

12a. What motivations did you have for using WCA?

- To have an autocomplete or reduce the amount of keystrokes I make
- To finish my programming tasks faster
- To skip needing to go online to find specific code snippets, programming syntax, or API calls I'm aware of, but can't remember
- To discover potential ways or starting points to write a solution to a problem I'm facing
- To find an edge case for my code I haven't considered
- My line management recommended I use it
- My colleagues recommended I use it
- I like to explore new tools
- It is my responsibility to try new IBM products
- I wanted to know more about how my work might change in the future

12b. Did you have any other reasons for using WCA? What were they?

- *Open-ended response*

A.13 Use of generated content

If the respondent indicated they used WCA within the past two weeks, they were asked these questions about how they made use of content generated by WCA.

13a. Within the past 2 weeks, how have you used content produced by WCA? Please check all that apply. Please respond based only on your usage of WCA within the past 2 weeks.

- Code snippets (excluding unit tests)
- Unit tests
- Natural language explanations
- Documentation

The items above were rated on the following scale: I included it in my code without modification, I included it in my code and made changes to it, I used it to give me new ideas, I used it to learn something new, I didn't make use of it

13b. Within the past 2 weeks, how did you decide when to use WCA versus completing a task yourself? Please respond based only on your usage of WCA within the past 2 weeks.

- *Open-ended response*

A.14 Impact on productivity

These scales were reproduced from Weisz et al. [53]. Each item was rated on a 7-point semantic differential scale.

14. Do you feel that WCA makes your work...

- More difficult / Easier
- Slower / Faster
- Of the worst quality / Of the best quality

A.15 In-depth use of generated content

The items in this section were rated on a 5-point scale: Never, Rarely, Sometimes, Often, Always.

15. How often do you take these actions to evaluate content generated by WCA?

- Quickly check the generated code for specific keywords or logic structures
- Compile, type check, lint, and/or use an in-IDE syntax checker
- Execute the generated code
- Examine details of the generated code's logic in depth
- Consult API documentation

A.16 Authorship

These items in this question were rated on a 4-point scale: Me, Both of us, WCA, I'm unsure.

16a. When using WCA, who do you consider to be the author of the code when...

- I paste WCA-generated code into my source file verbatim

- I paste WCA-generated code into my source file but then make edits to it
- I review code generated by WCA, but ultimately implement the functionality myself
- I implement an idea suggested by WCA

16b. How does WCA change your perceptions of your job role, responsibilities, and purpose as a developer?

- *Open-ended response*

A.17 Final comments

17a. Please describe your ideal vision for how WCA could help boost your productivity as a developer. Consider the following ideas: What additional features or capabilities does it need? How would it fit into your workflow? What kinds of tasks would you use it for and what kinds of tasks would you not use it for?

- *Open-ended response*

17b. Is there any other feedback you would like to provide about WCA or this survey?

- *Open-ended response*

B Respondents & participants

For survey respondents, we show the distributions of their years of experience as a professional software engineer in Figure 1a, tenure with IBM in Figure 1b, and geography in Figure 1c (survey questions 9a-9c).

C Motivations for using AI programming assistants

Table 2 shows a detailed comparison of motivations for using or not using AI programming assistants between our survey respondents and the software engineers who responded to the survey reported by Liang et al. [28] (survey question 12a). We include a number of additional motivations (A1-10) identified as important by our product management team. We also note that respondents were only asked to select which motivations explained their non-use of WCA rather than rating them on the 5-point scale of importance; we therefore only report the percentage of respondents who indicated each motivation for non-use. In addition, we did not include all motivations for non-use from Liang et al. [28] as some were not relevant for our enterprise context.

D Purposes of use of WCA

Figure 3a shows the distribution of purposes of use of WCA (survey question 2a). These data were reported by respondents who indicated they had used WCA within the prior two weeks (N=638).

E Distributions of productivity measures

Figure 2a shows detailed distributions of self-reported ratings of effort, quality of work, and speed (survey question 14). These items were rated on 7-point semantic differential scales, centered on 0. In Figure 2b, we show the distribution of self-efficacy scores. In Figure 2c, we show the distribution of quality scores.

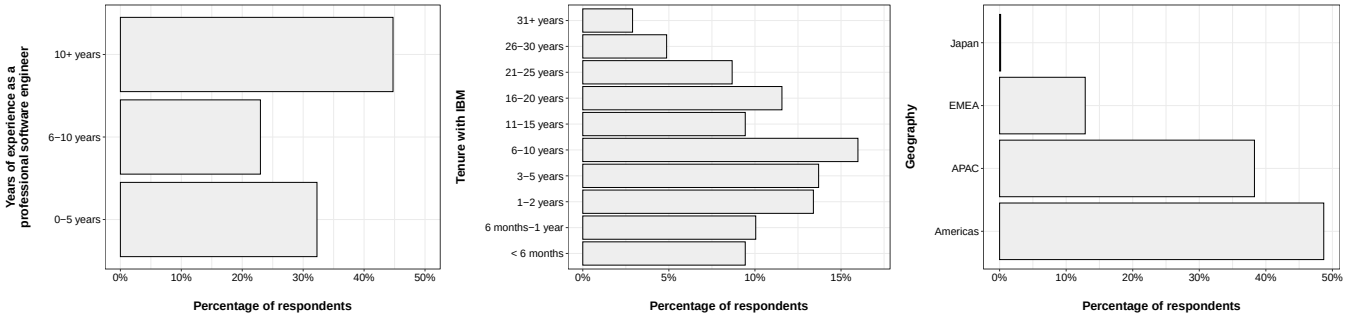
F Views on code authorship across co-creative scenarios

Figure 3b shows the distribution of respondents' views on code authorship across different co-creative scenarios (survey question 16a). Although the major trends align with our intuitions (e.g. when a party authors code, they are an author), we note that across all scenarios, some proportion of respondents felt a joint ownership with WCA.

Motivations		Liang et al. [28]	Our respondents
Motivations for use (Liang et al. [28])			
M1	To have an autocomplete or reduce the amount of keystrokes I make	86% 6.2%	47% 30%
M2	To finish my programming tasks faster	76% 12%	59% 21%
M3	To skip needing to go online to find specific code snippets, programming syntax, or API calls I'm aware of, but can't remember	68% 14%	64% 17%
M4	To discover potential ways or starting points to write a solution to a problem I'm facing	50% 24%	64% 17%
M5	To find an edge case for my code I haven't considered	36% 44%	48% 28%
Additional motivations for use			
A1	My colleagues recommended I use it	–	30% 39%
A2	My line management recommended I use it	–	60% 19%
A3	I like to explore new tools	–	68% 14%
A4	I wanted to know more about how my work might change in the future	–	64% 18%
A5	It is my responsibility to try new IBM products	–	64% 18%
Motivations for non-use (Liang et al. [28])			
M6	Code generation tools write code that doesn't meet functional or non-functional (e.g., security, performance) requirements that I need	54% 34%	14.3%
M7	It's hard to control code generation tools to get code that I want	48% 36%	14.3%
M8	I spend too much time debugging or modifying code written by code generation tools	38% 45%	14.3%
M9	I don't think code generation tools provide helpful suggestions	34% 46%	32.1%
M10	I don't want to use a tool that has access to my code	30% 51%	–
M11	I write and use proprietary code that code generation tools haven't seen before and don't generate	28% 59%	–
M12	To prevent potential intellectual property infringement	26% 66%	–
M13	I find the tool's suggestions too distracting	26% 51%	–
M14	I don't understand the code written by code generation tools	16% 76%	3.6%
M15	I don't want to use open-source code	10% 89%	–
Additional motivations for non-use			
A6	Code generated by WCA doesn't perform well enough for my needs	–	17.9%
A7	I don't understand the explanations generated by WCA	–	3.6%
A8	I have not done any code-related tasks in the past 2 weeks	–	25.0%
A9	I write and use code that WCA wasn't trained on which limits its ability to provide assistance	–	10.7%
A10	It is faster to do the work myself	–	39.3%

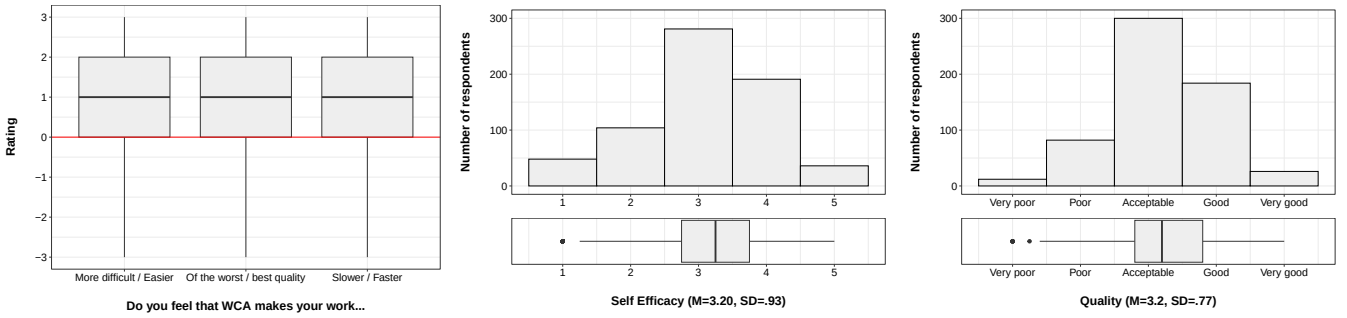
Very important Important Moderately important Slightly important Not important at all

Table 2: Comparison of motivations for using (M1-5) and not using (M6-15) AI programming assistants with Liang et al. [28], along with additional motivations (A1-10) examined in our survey. Percentages on the left-hand side (right-hand side) of each bar indicate the proportion of respondents who rated each motivation as "Very important" or "Important" ("Slightly important" or "Not important at all") on a 5-point scale. Only 28 of our 669 respondents (4.2%) indicated non-use of WCA within the prior two weeks.



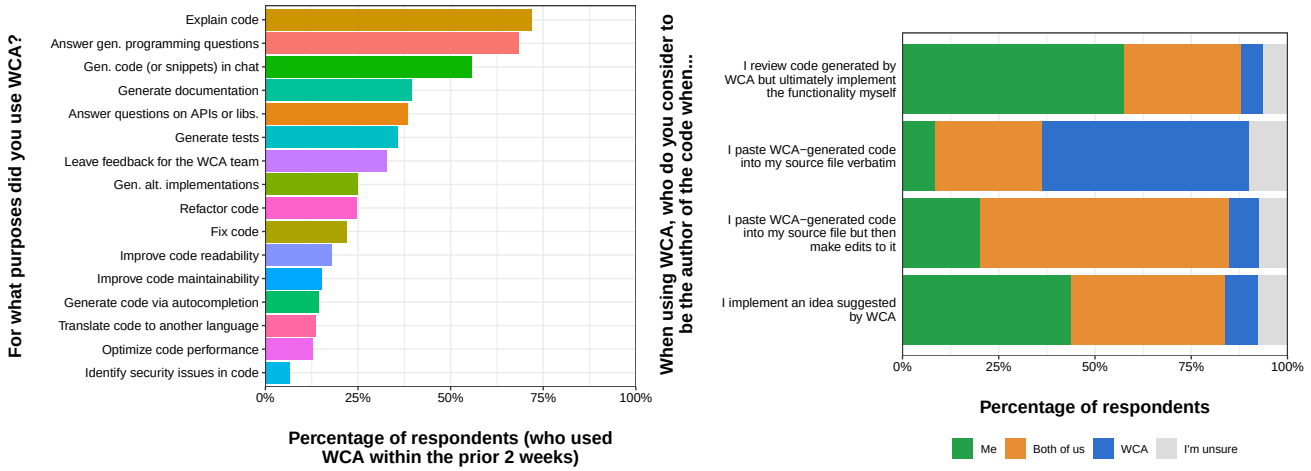
(a) (b) (c)

Figure 1: Survey respondent demographics: (a) years of experience as a professional software engineer, (b) tenure with IBM, and (c) geography.



(a) (b) (c)

Figure 2: Distributions of productivity and quality measures: (a) self-reported ratings of effort, quality of work, and speed on 7-point semantic differential scales (centered at 0), (b) distribution of self-efficacy scores, and (c) distribution of overall quality scores.



(a) Purposes of use of WCA for the prior two weeks. **(b) Views on code authorship across different co-creative scenarios.**

Figure 3: Distributions of purposes of use of WCA and views on code authorship.