An Engineering Perspective on Writing Assistants for Productivity and Creative Code

Ambar Murillo and Sarah D'Angelo
ambarm@google.com,sdangelo@google.com
Google

1 INTRODUCTION

Software developers write code nearly everyday, ranging from simple straightforward tasks to challenging and creative tasks. As we have seen across domains, AI/ML based code writing assistants are on the rise in the field of computer science. We refer to them as code generation tools or ML enhanced software developing tooling; and it is changing the way developers write code [2, 10]. As we think about how to design and measure the impact of intelligent writing assistants, the approaches used in software engineering and the considerations unique to writing code can provide a different and complementary perspective for the workshop. In this paper, we propose a focus on two themes: (1) measuring the impact of writing assistants and (2) how code writing assistants are changing the way engineers write code. In our discussion of these topics, we outline approaches used in software engineering, and how the disciplines of prose writing and code writing can learn from each other. We aim to contribute to the development of a taxonomy of writing assistants that includes possible methods of measurement and considers factors unique to each domain (e.g. prose or code).

2 MEASURING THE IMPACT OF WRITING ASSISTANTS IN SOFTWARE ENGINEERING

Many companies have recently developed publicly available code generation tools including Copilot from Github [6], AlphaCode from Alphabet's DeepMind [9], and CodeWhisperer from Amazon [1]; as well as internal tools from companies such as Meta [2] and Google [12]. To better understand the impact of these tools, a number of metrics have been leveraged to understand how developers interact with code writing assistants and how it impacts the code that is written, including:

Behavioral metrics: These include daily completions per user (DCPU), the raw number of accepted suggestions per developer per day [15], coding iteration time, persistence (the ratio of accepted completions unchanged) [16] and the percentage of new code generated from accepting ML code completion suggestions [12]. These measures provide feedback on performance of the ML model, its impact to workflows and output, and can be applied to a wide range of code writing assistants, beyond the coding use case.

Attitudinal measures: These include user sentiment such as perceived utility [4] and perceived productivity [13].

Some approaches have combined both logs-based measures with attitudinal measures to better understand interaction patterns. For example, Mozannar et al [10] developed a taxonomy for interactions with code writing assistants (Copilot) by asking participants to label logs based metrics to capture their intent when working with the code writing assistant. Other approaches have used both log-based and attitudinal measures to identify where behaviors and perceptions are aligned or conflict. For example, researchers showed that developers feel more productive when using ML-based assistance, even if they aren't always faster at producing code [13].

While these metrics give us some insight into how code writing assistants are changing the way engineers write code, they are only part of the story. Recent research has started to explore how code writing assistants are altering workflows and mindsets.

3 HOW CODE WRITING ASSISTANTS ARE CHANGING THE WAY DEVELOPERS WRITE CODE

ML enhancement in software development tooling aims to improve the efficiency and productivity of software developers by 1) reducing developer workloads; 2) reducing task time; and 3) mitigating human errors. The introduction of code writing assistants into developer tooling is also starting to shift the way engineers think about writing code. For example, Barke et al [3] have looked at how developers interact with Copilot, and identified two modes of working with ML-based assistance: acceleration (where the developer used Copilot to execute planned actions, staying in flow, with a focus on accelerating their development) and exploration (where the developer used Copilot to help plan next steps and explore possible paths forward). Other research has also looked at interactions and initial experiences with Copilot, describing how the process of writing code has changed as a result of Copilot, and is moving toward reviewing code [5]. This comes with new challenges, for example that debugging code written by ML enhanced developer tooling may be harder than debugging code written by the engineer [11].

These studies highlight the shifts in how code writing assistants are fundamentally changing how engineers write code. However, this is just the beginning, there is more to understand about how engineering workflows are impacted by assistants. Under explored areas at the intersection of human-AI collaboration and engineering are creativity and collaboration, which are starting to be explored in recent prose writing research. For example, Ippolito et al, [7] examined how participants (experienced authors) interacted with a writing assistant for creative writing tasks, finding that the authors enjoyed brainstorming and adding details aided by the writing assistant, but did not want to "offload the creative process" to it. The authors also highlight challenges of using writing assistants for creative writing, for example the assistants have trouble maintaining a style and voice, and suggestions often reverted to "tropes and repetition". Additionally, Lee et al, [8] explored how GPT-3, a large language model (LLM), can be evaluated as a writing collaborator and discuss opportunities for using metrics to understand how writers use writing assistants.

In preparation for a discussion on how software engineering can leverage approaches and insights from other forms of writing, particularly in the realm of creativity, it is important to consider how coding differs from prose writing. For example, code has to follow a more rigid structure, it has also been described as more "brittle" compared to natural languages, since small modifications could have large implications, and finally it should successfully build and run, without introducing bugs [14]. Acknowledging these differences, recent research has started to look at human-AI collaboration in the context of code translation [14].

Similar to grammatical correctness, structural requirements and dependencies for code writing may influence how we think about creativity in code writing assistants for developers. Approaches on how to further explore the influence of code writing assistants on creativity and novelty in the context of software engineering could help inform future research and design.

4 CONCLUSION

In this paper, we propose measurements for evaluating the impact of writing assistants and how they are changing the way we write, from the perspective of software engineering. If invited to attend this workshop, we are well suited to contribute perspectives on code writing assistants in the context of software engineering: its impact on coding and how to measure this impact. We would benefit from discussing how writing assistants have changed other disciplines: how they have been used and evaluated in . As the workshop involves a collaboration creation of a taxonomy of writing assistants, we believe including measurement approaches would be a valuable addition. Additionally, exploring the similarities and differences in prose writing compared to code, will spark ideas for future research and collaboration.

REFERENCES

[1] Amazon. 2022. ML-powered coding companion for developers - Amazon CodeWhisperer Features. https://aws.amazon.com/codewhisperer/features/

[2] Johannes Bader, Sonia Seohyun Kim, Frank Sifei Luan, Satish Chandra, and Erik Meijer. 2021. AI in Software Engineering at Facebook. *IEEE Software* 38, 4 (2021), 52–61. https://doi.org/10.1109/MS.2021.3061664

[3] Shraddha Barke, Michael B James, and Nadia Polikarpova. 2022. Grounded copilot: How programmers interact with code-generating models. *arXiv preprint arXiv:2206.15000* (2022).

[4] Moritz Beller, Hongyu Li, Vivek Nair, Vijayaraghavan Murali, Imad Ahmad, Jürgen Cito, Drew Carlson, Ari Aye, and Wes Dyer. 2022. Learning to Learn to Predict Performance Regressions in Production at Meta. *arXiv preprint arXiv:2208.04351* (2022).

[5] Christian Bird, Denae Ford, Thomas Zimmermann, Nicole Forsgren, Eirini Kalliamvakou, Travis Lowdermilk, and Idan Gazit. 2023. Taking Flight with Copilot: Early Insights and Opportunities of AI-Powered Pair-Programming Tools. *Queue* 20, 6 (jan 2023), 35–57. https://doi.org/10.1145/3582083

[6] Nat Friedman. 2021. Introducing github copilot: Your AI pair programmer. https://github.blog/2021-06-29-introducing-github-copilot-ai-pair-programmer/

[7] Daphne Ippolito, Ann Yuan, Andy Coenen, and Sehmon Burnam. 2022. Creative Writing with an AI-Powered Writing Assistant: Perspectives from Professional Writers. *arXiv preprint arXiv:2211.05030* (2022).

[8] Mina Lee, Percy Liang, and Qian Yang. 2022. Coauthor: Designing a human-ai collaborative writing dataset for exploring language model capabilities. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. 1–19.

[9] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. 2022. Competition-level code generation with alphacode. *Science* 378, 6624 (2022), 1092–1097.

[10] Hussein Mozannar, Gagan Bansal, Adam Fourney, and Eric Horvitz. 2022. Reading Between the Lines: Modeling User Behavior and Costs in AI-Assisted Programming. *arXiv preprint arXiv:2210.14306* (2022).

[11] Advait Sarkar, Andrew D Gordon, Carina Negreanu, Christian Poelitz, Sruti Srinivasa Ragavan, and Ben Zorn. 2022. What is it like to program with artificial intelligence? *arXiv preprint arXiv:2208.06213* (2022).

[12] Maxim Tabachnyk and Stoyan Nikolov. 2022. ML-enhanced code completion improves developer productivity. https://ai.googleblog.com/2022/07/ml-enhanced-code-completion-improves.html

[13] 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.

[14] 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 *26th International Conference on Intelligent User Interfaces (IUI '21)*. Association for Computing Machinery, New York, NY, USA, 402–412. https://doi.org/10.1145/3397481.3450656

[15] Wen Zhou, Seohyun Kim, Vijayaraghavan Murali, and Gareth Ari Aye. 2022. Improving Code Autocompletion with Transfer Learning. In *Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP '22)*. Association for Computing Machinery, New York, NY, USA, 161–162. https://doi.org/10.1145/3510457.3513061

[16] 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.