Is GitHub Copilot a Substitute for Human Pair-programming? An Empirical Study

Saki Imai
simai24@colby.edu
Colby College
Waterville, Maine, USA

ABSTRACT

This empirical study investigates the effectiveness of pair programming with GitHub Copilot in comparison to human pair-programming. Through an experiment with 21 participants we focus on code productivity and code quality. For experimental design, a participant was given a project to code, under three conditions presented in a randomized order. The conditions are pair-programming with Copilot, human pair-programming as a driver, and as a navigator. The codes generated from the three trials were analyzed to determine how many lines of code on average were added in each condition and how many lines of code on average were removed in the subsequent stage. The former measures the productivity of each condition while the latter measures the quality of the produced code. The results suggest that although Copilot increases productivity as measured by lines of code added, the quality of code produced is inferior by having more lines of code deleted in the subsequent trial.

CCS CONCEPTS

• **Software and its engineering** → **Development frameworks and environments**; • **Human-centered computing** → *Collaborative and social computing systems and tools.*

KEYWORDS

GitHub, Copilot, Software Development, AI

ACM Reference Format:
Saki Imai. 2022. Is GitHub Copilot a Substitute for Human Pair-programming? An Empirical Study. In *44th International Conference on Software Engineering Companion (ICSE '22 Companion), May 21–29, 2022, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3510454.3522684

1 RESEARCH PROBLEM AND MOTIVATION

GitHub Copilot is a software development tool that offers code generation of lines, code chunks, or even entire programs based on existing code and comments [10]. Copilot is marketed as a substitute for pair-programming, a software development practice where two programmers collaboratively write a single piece of code.

If GitHub Copilot could produce equivalent advantages found in human pair-programming, then adopting the practice of pair-programming with GitHub Copilot would lead to a more productive and higher quality software development without acquiring additional costs of adding a second programmer. While Vinit Shahdeo, a software engineer at Postman, said Copilot is "going to increase developer's efficiency by reducing development time and suggesting better alternatives", technical blogger Ray Villalobos states that it is hard to get a useful result and that he needs to retype comments to get a productive piece of code [2]. Although there are claims that these AI tools make software development more productive and they could even substitute human pair-programmers, we have not seen an empirical study to verify if AI tools in software development are more productive and give higher quality code. In this paper, we focus on the issue of productivity and code quality when using GitHub Copilot in software development. We designed a dedicated empirical experiment to compare AI with human participants in a natural software development environment. Through code analysis, we aim to answer our two central research questions focusing on measuring productivity and code quality with GitHub Copilot.

2 BACKGROUND AND RELATED WORK

We recognize two major themes in the previous works that have been done in this field. The first is the use of AI in software development. Many studies have shown that the use of AI assists with software development. For instance, one study used a transformer-based model reported accuracy of up to 69% in predicting tokens when code tokens were masked [4]. Another study using large language models reported that AI could repair 100% of handcrafted security bugs in addition to 58% of historical bugs in open-source projects [8]. Moreover, a trained GPT language model has been exhibited to solve 70.2% of problems with 100 training samples per problem [3], and is also capable of repairing bugs in code [9]. One study predicted defects with 87% accuracy, decreased inspection effort by 72%, and reduced post-release defects by 44% [11].

The second theme focuses on the study of software development environments, where empirical experimentation of how people write code gives us insights into how to enhance these tools and to possibly discover the best practice of software development [6]. There have been studies on how professional developers comprehend software to understand how software development should be done, such as how programmers refactor while validating other programmers [7], and how implementation of task context for the Eclipse development environment improved productivity of programmers [5]. We recognize that the study of AI tools in software development has not been studied empirically with a dedicated experiment.

3 APPROACH AND UNIQUENESS

In this research, we aim to study GitHub Copilot empirically in a natural software development environment (VS Code IDE). Hence, the research questions to be addressed in this study are as follows: (RQ1) Is there an advantage in productivity while using GitHub Copilot as compared to a human pair programmer? (RQ2) What is the quality of code written with Copilot in comparison to human pair programmers?

In pair programming, two programmers collaboratively work on the same code (typically on the same computer). Each programmer periodically switches between two roles, a driver or navigator. The driver controls the mouse and keyboard and writes code while the navigator observes the driver's work and critically thinks about defects, structural issues, and alternative solutions, while looking at a larger picture [1].

Using GitHub Copilot as a second programmer, we compare code when a participant is pair programming with a human programmer versus Copilot. Twenty-one participants who have taken at least one programming course worked on developing text-based minesweeper game in Python. None of the participants had implemented this game before, and the participants familiarized themselves with the rules by playing this game prior to the development task. The development task was done under three conditions. The conditions are pair programming with Copilot; pair programming with another human experimenter as a driver, and pair programming with another human experimenter as a navigator. The time allocated for is 20 minutes for Copilot, 10 minutes as a driver, and 10 minutes as a navigator (20 minutes total with a human pair). The order of these conditions were randomized to prevent the experiment effect. During the experiment, eye movement is recorded to measure the difference between having Copilot as a collaborator in comparison to a human programmer.
The analysis of the produced code is done by using the ndiff function from difflib[1]. This is used to compare the number of added lines to the code and number of deleted lines to the code after each trial, normalized by the duration of the trial.

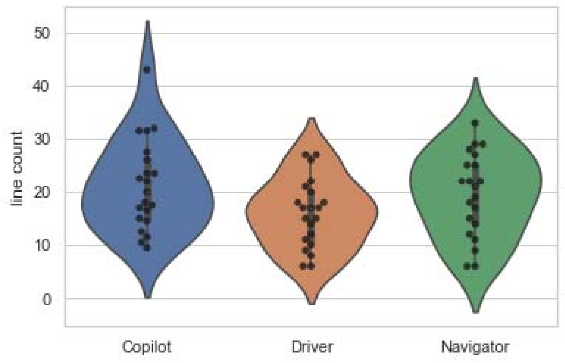

Figure 1: Number of lines added to a code under three different conditions.

[1]https://docs.python.org/3/library/difflib.html

4 RESULTS

To answer our research questions, code productivity in RQ1 is assessed by comparing the number of added lines to the code, and code quality in RQ2 is analyzed by comparing the number of lines deleted in the subsequent trial. Deletion is an indication of low quality code.

The result of RQ1 is shown in Figure 1, where we can see that the Copilot condition produced the highest maximum and mean additions to lines of code. The maximum number of lines written in the trial with Copilot was 43 while the code written as a driver and navigator were 27 and 33 respectively. The minimum lines of code added was 9.5 for Copilot and 6 for both driver and navigator. These results suggest higher productivity during pair-programming with Copilot versus human pair-programmers.

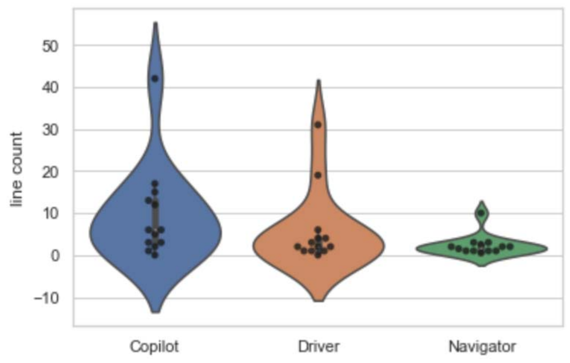

Figure 2: Number of lines of code deleted in a trial subsequent to three different conditions.

To answer RQ2, we counted the number of deleted lines in the following trial and normalize the count by the trial duration. For this, the line counts for the last condition were excluded since there was no trial subsequent to that where low quality code can be removed. The maximum lines of code deleted after the Copilot trial was 42 while the lines of code deleted after the driver and navigator trial were lower with 31 and 10, respectively. Figure 2 also shows that the deleted line count in the following trial was higher for Copilot than the other two conditions. Hence, our result suggests that the code generated with Copilot has, on average, lower quality than that produced by human pair-programmers.

5 CONTRIBUTIONS

Our results suggest that although programming with Copilot helps generate more lines of code than human pair-programming in the same period of time, the quality of code generated by Copilot appears to be lower. This result seems to suggest that pair-programming with Copilot does not match the profile of human pair-programming.
We are still in the process of collecting experiment data and analyzing the eye-tracking data that have been recorded throughout the experiment. With the eye-tracking data, we are trying to compare how programmer inspect the code generated by AI to that by human pair-programmer. Our hypothesis is that the overconfidence of AI tools leads to less inspection of code generated by Copilot.

REFERENCES

[1] Ritchie Schacher Adam Archer and Scott Will. [n.d.]. *Program in Pairs.* Retrieved December 31, 2021 from https://www.ibm.com/garage/method/practices/code/practice_pair_programming/

[2] Scott Carey. 2021. *Developers react to GitHub Copilot.* Retrieved December 31, 2021 from https://www.infoworld.com/article/3624688/developers-react-to-github-copilot.html

[3] 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).

[4] Matteo Ciniselli, Nathan Cooper, Luca Pascarella, Antonio Mastropaolo, Emad Aghajani, Denys Poshyvanyk, Massimiliano Di Penta, and Gabriele Bavota. 2021. An Empirical Study on the Usage of Transformer Models for Code Completion. *IEEE Transactions on Software Engineering* (2021).

[5] Mik Kersten and Gail C Murphy. 2006. Using task context to improve programmer productivity. In *Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering*. 1–11.

[6] Gail C Murphy, Mik Kersten, and Leah Findlater. 2006. How are Java software developers using the Elipse IDE? *IEEE software* 23, 4 (2006), 76–83.

[7] Emerson Murphy-Hill, Chris Parnin, and Andrew P Black. 2011. How we refactor, and how we know it. *IEEE Transactions on Software Engineering* 38, 1 (2011), 5–18.

[8] Hammond Pearce, Benjamin Tan, Baleegh Ahmad, Ramesh Karri, and Brendan Dolan-Gavitt. 2021. Can OpenAI Codex and Other Large Language Models Help Us Fix Security Bugs? *arXiv preprint arXiv:2112.02125* (2021).

[9] Julian Aron Prenner and Romain Robbes. 2021. Automatic Program Repair with OpenAI's Codex: Evaluating QuixBugs. *arXiv preprint arXiv:2111.03922* (2021).

[10] Dominik Sobania, Martin Briesch, and Franz Rothlauf. 2021. Choose Your Programming Copilot: A Comparison of the Program Synthesis Performance of GitHub Copilot and Genetic Programming. *arXiv preprint arXiv:2111.07875* (2021).

[11] Ayse Tosun, Ayse Bener, and Resat Kale. 2010. Ai-based software defect predictors: Applications and benefits in a case study. In *Twenty-Second IAAI Conference.*