blueoptima

REPORT

The Impact of Generative AI on Software Developer Performance

blueoptima

The Impact of Generative AI on Software Developer Performance

Abstract

This study is the largest and most comprehensive to date investigating the impact of Generative AI (GenAI) tools on software developer performance, addressing productivity, code quality, and adoption patterns. Using a mixed-method approach combining quasi-experimental design and Code Author Detection (CAD) analysis, we examined 218,354 professional software developers working in an enterprise setting, covering about 880 million commits extracted from mid-2022 to mid-2024. Despite the large sample, only 2,062 developers consistently committed GenAI-authored code, demonstrating that very few developers exploit GenAI such that significant rework of the code provided is not required before committing it to source code repositories. Employing two independent methodologies to control for GenAI usage effects, we demonstrate that GenAI tools enhance developer productivity by approximately 4%, while generally maintaining code quality. Remarkably, developers with moderate GenAI usage emerged as the highest overall performers, suggesting an optimal balance between AI assistance and human expertise. The study also uncovered unexpected adoption patterns, with 64% of licensed GenAI users employing GenAI before officially being granted licences. These results show that GenAI use does not significantly reduce code quality and suggest potential for GenAI to improve software development productivity. However, the low commit rate of GenAI authored code unaltered by human software engineers underscores the need for further research into implementation strategies and long-term impacts of GenAI in software engineering.

"Why are you still here, I thought we got a Copilot subscription?"

Introduction

The software development industry has generated a great deal of anticipation and expectation regarding the integration of Generative AI (GenAI) technologies. These advancements, particularly through Large Language Models (LLMs), present the possibility of significantly enhancing developer productivity, improving code quality, and reducing costs by automating routine and repetitive tasks. GenAI tools can generate code snippets, complete code blocks, and refactor existing codebases based on high-level instructions, which could streamline the development process.

Historically, innovations such as compilers in the 1950s and Integrated Development Environments (IDEs) in the 1990s have substantially boosted developer efficiency and code reliability (Levy & Prlić, 2022). Similarly, GenAI is expected to reduce the cognitive load on developers, enabling them to focus on more complex and strategic tasks (Forsgren, Humble, & Kim, 2018). However, integrating GenAI tools also introduces challenges, such as the need for adequate training, the risk of over-reliance on AI-generated code, and ensuring the security and quality of the code authored by GenAI (Guan et al., 2023).

Before getting into a detailed account of the impact of GenAI on software developer performance, it is important to understand the broader context of AI's impact on productivity. Recent studies and analyses suggest that, despite significant investments and widespread adoption claims, the economic impact of AI technologies, including GenAI, remains limited and difficult to quantify. For instance, a report by The Economist highlights that the anticipated revolution in productivity driven by AI technologies has not yet materialised in general economic measures, raising questions about the real-world effectiveness of these tools (The Economist, 2024).

The advent of Generative AI (GenAI) has already significantly impacted various industries, one of the industries likely to be impacted significantly appears to be software development. GenAI tools, such as Microsoft Copilot, leverage machine learning to assist developers by automating the initial drafting of code, providing code explanations, and identifying potential bugs in source code or other issues. While some industry participants, particularly those selling Generative AI solutions, suggest that the integration of AI into the software development life cycle has the potential to substantially enhance productivity and code quality. Other industry commentators suggest that the inclusion of GenAI into software development workflows may impair software developer performance, often anticipating that the review and correction of inappropriate and or poor quality code will ultimately impair progress. However, the actual impact of these tools on developer performance remains a subject of empirical investigation.

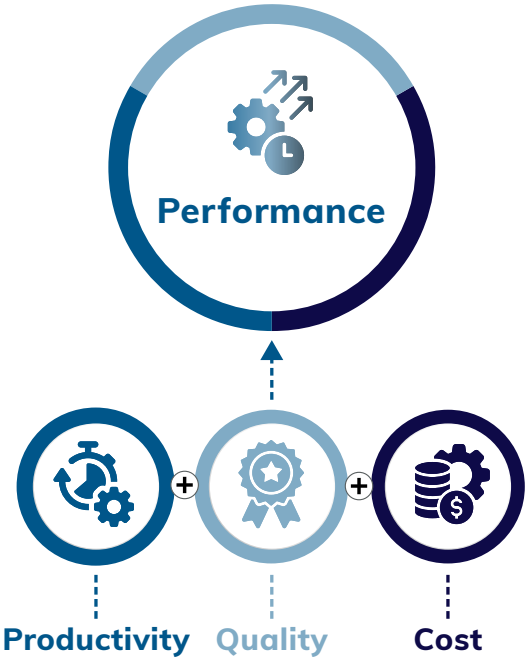
This study is the largest and most comprehensive to date investigating the impact of Generative AI (GenAI) tools on software developer performance

Previous studies have demonstrated mixed results regarding the impact of GenAI on software development. For instance Peng et al. (2023) found that developers using Copilot completed tasks 55.8% faster than a control group. Similarly, Almeida et al (2024) reported that AI tools improved the detection of code smells in code review by 40% and faster than manual review, resulting in higher overall code quality. Chatterjee et al. (2024) use artificial coding tasks to evaluate the impact on developer performance in an enterprise setting and generally describe improved performance and satisfaction though decline to quantify this specifically. However, these studies primarily relied on self-reported data and lacked rigorous experimental controls, making it difficult to generalise the findings.

The integration of AI tools in software development has raised concerns about potential trade-offs between productivity and code quality. Some researchers have suggested that while AI might increase the speed of code production, it could potentially lead to lower code quality or increased technical debt (Sobreira et al., 2023; Lwakatare et al., 2022). This concern stems from the possibility that developers might rely too heavily on AI-generated code without fully understanding or optimising it, or that the pressure to deliver code quickly might overshadow quality considerations. However, others argue that AI tools, when used appropriately, could enhance both productivity and quality by allowing developers to focus on higher-level design and problem-solving tasks (Rahman et al., 2023). This study

provides empirical evidence to address this debate and assess the actual impact of GenAI tools on both productivity and code quality in real-world development environments.

A fundamental shortcoming of prior research is an incomplete and inadequate account of software developer performance. This paper addresses this shortcoming by evaluating software development performance as comprising three components: productivity, quality, and cost. These are the primary considerations of any engineering endeavour, and software engineering is no exception to the challenge of simultaneously optimising these three fundamental dimensions of performance.



The measure of Productivity, Coding Effort, is a measure of the intellectual effort expended by a developer delivering a single change into the source code of an application. Coding Effort is calculated by statistically evaluating every source code change made by developers in terms of up to 36 static source code metrics measuring various aspects of Volume, Complexity, and Interrelatedness while considering the context worked in (e.g. a complex legacy software component or a brand new project). Coding Effort is evaluated based on the changes committed into version control systems on a per commit per file basis, which is called Actual Coding Effort (ACE). As a measure of Productivity, Billable Coding Effort per developer per day is a measure of the average amount of Coding Effort

We analyse the impact of GenAI on software developer performance, focusing on Productivity (measured as Billable Coding Effort per Day, BCE/ Day) and Quality (measured as the percentage of Aberrant Coding Effort, % Aberrant BCE)

an individual developer delivers per day To obtain the target variable of this paper (BCE/day), Billable Coding Effort is adjusted for stored changes and prorated across working days (i.e. adjusted for weekends and holidays) to get the average effort per day, or BCE/day. The range for BCE/day ranges from 0 to 5.5. This means that the maximum coding effort a developer can obtain on a single day is 5.5 hours. This is a populational maximum, based on empirical studies, that accounts for the time that top decile developers would spend on "at keyboard programming" aspects of their role. This excludes meetings, breaks, and non-coding work.

The measure of Quality, Aberrant Coding Effort, is a measure of the quality, more specifically maintainability, of source code. The aberrancy of source code provides an objective account of how easy it is for a developer who is naive to the code to reach a level of understanding of the code so that they can extend, alter, or improve it. It is calculated by evaluating the proportion of code that is aberrant relative to the codebase in which it sits across more than 20 static source code metrics. Code is flagged as aberrant when it violates thresholds that have been benchmarked across an enterprise's software estate and BlueOptima's Global Benchmarks. The percentage represents the ratio of coding effort devoted to unmaintainable changes so a lower percentage is more desirable. It considers the maintainability of a file before changes are made, so no penalty is given for changes in an already unmaintainable file unless they push a static metric further past the threshold of un-maintainability.

For the purposes of the relatively short duration of this study, Cost, in the context of software development performance, is the fully loaded cost of employing software developers. This is assumed to be a constant as individual developers are only considered if they are continuously employed by the software development organisation for the duration of the study.

This study also addresses prior research's shortcomings in isolating the impact of using GenAI versus control groups who do not have access to the technology. We employ a robust quasi-experimental design and leverage BlueOptima's Code Author Detection (CAD) capabilities. We analyse the impact of GenAI on software developer performance, focusing on Productivity (measured as Billable Coding Effort per Day, BCE/Day) and Quality (measured as the percentage of Aberrant Coding Effort, % Aberrant BCE).

The study is structured around four key questions:

1

"What are the effects of GenAI use on developer productivity and quality?"

Impact of GenAI Use on Selected Developers in Selected Teams: We use a quasi-experimental prospective design to compare the performance of developers identified as being granted access to GenAI tools (Experimental Group) with those who were not (Control Group). Our aim is to determine whether GenAI use leads to observable differences in productivity and quality.

2

"Were developers accurately allocated to GenAI and control groups?"

Control and Representativeness of Selected Developers in Teams Approach: By applying an automated approach to detecting the use of GenAI by those developers allocated to the experimental group, we investigate the extent to which the developers were appropriately allocated to these groups. This includes assessing whether the selected developers commit GenAI-authored code before the date at which they were allocated GenAI tool licences or if they commit GenAI-authored code at all

3

"What are the trends in GenAI usage and its impact on productivity and quality across a large developer population?"

Observations Across Large Developer Populations: We extend our analysis to a larger cohort of developers and evaluate those who have, through the course of their normal work commenced using GenAI in a post hoc quasi-experimental design. This broader analysis helps us understand the general trends in GenAI usage and its impact on the same productivity and quality metrics across a much larger and more diverse developer population.

4

"How do different levels of GenAI usage (e.g. high, low, none) influence developer performance?"

Drivers of Differences in Productivity and Quality: We then explore the relationships between GenAI usage levels and developer performance. Specifically, we examine the relationship between high, low, and no GenAI usage on performance and how these insights can inform more effective utilisation of AI tools in software engineering.

This study, which is the most comprehensive and large-scale study of its nature that we are aware of, allows us to provide a robust and detailed understanding of the impact of GenAI on software development, highlighting both its potential benefits and the complexities involved in its implementation.

Method

Study Design

This study employs a mixed-method approach, utilising both quasi-experimental design and BlueOptima's Code Author Detection (CAD) analysis to evaluate the impact of Generative AI (GenAI) on software developer performance. BlueOptima's Code Author Detection is a Machine Learning technology designed and developed by BlueOptima to detect AI-Contributed source code. This technology identifies code, at the method level, that has been authored by Machine Learning techniques, such as Generative AI (e.g. OpenAI ChatGPT, Meta CodeLlama, Google Gemini etc.), and committed to version control repositories without material editing from the committing developer. CAD does not capture IDE-based "autocomplete" contributions of ML/AI in cases where developers subsequently adjust and edit the autocompleted code making it stylistically and structurally unrecognisable as ML authored code. CAD does not seek to identify this usage of Machine Learning generated code as it is a less impactful assistive use case rather than a direct replacement of human developer effort that is not radically dissimilar to existing code completion technologies. See Appendix A for examples of the types of change that makes GenAI code Human Authored.

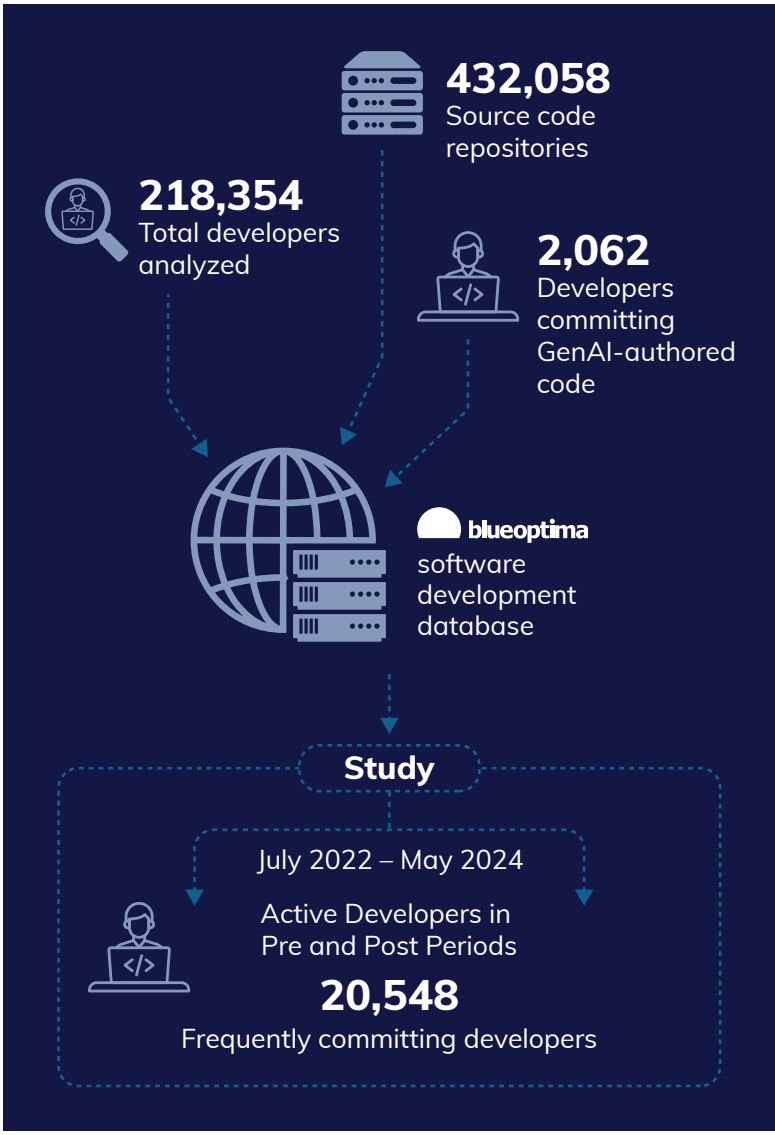
The analysis is conducted in two parts:

1. Licence-based Group Comparison: We use ANOVA to compare developers officially granted access to GenAI tools (Experimental Group) with those who were not (Control Group) within teams of software engineers.
2. CAD-based Group Comparison: We use BlueOptima's CAD capabilities to independently detect AI-authored code and assess the impact of GenAI usage on a larger population of developers who have used GenAI and compare them to those that have not over the same period of time using ANOVA.

Participants

Licence-based Group Comparison: This method involved identifying a group of developers to be allocated GenAI licences and identifying a control group corresponding with this selection of GenAI users.

Experimental Group (EG-ALL): The developers (N=3268) included in the study were identified by those that manage them in their software development organisation, in some cases they may have volunteered to receive the GenAI licences which were then approved by those managing the GenAI evaluation in their organisation. These developers belong to 317 software engineering teams within 28 Organisations across 10 Enterprises.



Control Group (CG-ACT-DEV): Active developers matching the activity criteria of EG-ALL but without GenAI tool licences (N = 3168) were sampled from 418 teams of software engineers across 40 Organisations within the same 10 Enterprises as the Experimental Group (EG-ALL) developers.

CAD-based Group Comparison: CAD is run over the activity of 218,354 professional software developers from July 2022 to May 2024 across

432,058 source code repositories operated by the software development organisations within 32 commercial enterprises. This population was reduced to 77,338 to include those developers that had activity in both the pre-treatment and post-treatment from their first ML-authored commit date to May 2024. This ensures that we are considering developers who have a tenure of activity over duration of the comparison. To further refine this population to more broadly active developers they were required to have committed for at least 6 months in each period, and to be active within 45 days of their ML-authored commit date. This refinement resulted in 47,881 developers in the pre-period and 28,525 in the post-period.

Next, only those developers who met the above criteria in both periods were selected, reducing the final population to 20,548 frequently committing developers. While this is a significant reduction in developer population given the initial activity of 218,354 developers, this allows us to be confident of the active development role of the final 9.4% of these developers on the relevant codebases over the relevant time periods.

Finally, CAD was run across the entire original population and identified 2,062 developers who had committed GenAI authored code within this refined group, allowing us to categorise the remaining 18,486 developers of the final 20,548 developers as the Zero-AI group. It is important to point out that unlike the Licence-based group, we cannot know the extent to which this group of 20,548 developers have had GenAI tools made available to them. While a number of them may have used the various free offerings available on the internet and others may have been officially permitted to use GenAI and been granted licences, others may have been prohibited from using GenAI as a matter of corporate policy.

To further refine those developers using GenAI the percentage of ML-authored Actual Coding Effort (ACE) was computed for the CAD developers to assess their degree of GenAI usage. Developers with an ML-authored ACE percentage higher than the median value were categorised as High AI-Contributing Developers, while those with a lower percentage were categorised as Low AI-Contributing Developers. This split the CAD group into two equal categories of 1,031 developers each.

Thus, three distinct developer groups were formed for the analysis as set out in the table below.

Developer Group	Description	Sample Size (N)
High AI-Contributing Developers	Developers with above average levels of AI-generated code in their commits	1031
Low AI-Contributing Developers	Developers with below average levels of AI-generated code in their commits	1031
No AI-Contributing Developers	Developers not committing any AI-generated code	18,486

Figure 1: This table shows the sample sizes used in the CAD-based Group Comparison

blueoptima

Identifying Pre and Post GenAI Usage

The validity of both approaches in isolating the impact of GenAI is to be able to accurately isolate the point at which developers commenced using GenAI. Below, we describe the point at which GenAI is used by the treatment group and how we have selected an appropriate time frame of observation for the control group.

Licence-based Group

The approach to identifying the transition point for the Licence-based Group comparison is straightforward in that it involves taking the date at which a particular subgroup or team were allocated their licences. Once this is done the comparison is made to members of the same subgroup or team who were not allocated licences as a control group. The time-frames considered are set out below.

Experimental Group (EG-ALL): 6 months before and after the date of licence allocation for a developer in this group, respectively.

Control Group (CG-ACT-DEV): Since this group was not provided a licence to Gen AI, this date was approximated using the median date of licence allocation among the EG-ALL developers. This ensures a fair comparison of the CG-ACT-DEV and EG-ALL groups. 6 months before and after this median date is the Pre and Post period for the CG-ACT-DEV group.

CAD-based Group Comparison

BlueOptima's CAD capabilities observed the initial usage of Gen-AI code in January 2023. To assess the pre-Gen-AI performance of these developers, a six-month time period from July 2022 to December 2022 would be considered the pre-period. July 2022 is the lower bound for performance assessment in the pre-time frame for all three categories of developers (High, Low, and No-AI), and May 2024 is the upper bound. Each developer is considered within this upper and lower bound as follows.

Developer Group	Pre-treatment Period	Post-treatment Period
High AI-Contributing Developers	From July 2022 to the date of the first ML-authored commit detected through CAD for a developer	From the first ML-authored commit date to May 2024
Low AI-Contributing Developers	From July 2022 to the date of the first ML-authored commit detected through CAD for a developer	From the first ML-authored commit date to May 2024
No AI-Contributing Developers	Date is approximated as the median of the first ML-authored commit date across the High and Low-AI developer population. So July 2022 to this median date would be the pre-period for this group	From the median ML-authored commit date for this group to May 2024

Figure 2: This table defines the different groups of developers using tn the CAD-based Group comparison

blueoptima

To keep the Licence-based Group Comparison ANOVA comparable with the CAD-based Group Comparison ANOVA the High AI-Contributing and Low AI-Contributing are combined for this analysis.

Statistical Analysis

1. Quasi-Experimental Design ANOVA: We conducted a two-way ANOVA to assess the main and interaction effects of group (Experimental vs. Control) and timeframe (Pre vs. Post) on productivity and quality metrics.

Productivity				Quality		
Developer Group	Pre BCE/ Day	Post BCE/ Day	% Change	Pre Aberrancy %	Post Aberrancy %	% Change
ML	2.32	2.42	+4.09%	5.95	5.96	+0.09%
Zero-AI	1.92	1.89	-1.81%	6.28	6.33	+0.89%
CG-ACT-DEV	2.12	2.01	-5.12%	6.393	6.390	-0.05%
EG-ALL	2.01	2.09	+3.99%	6.24	6.08	-2.63%

Figure 3: Productivity and quality metrics compared across 4 developer groups: ML – developers with GenAI access who made changes identified as GenAI, Zero-AI – developers with GenAI access who made changes identified as GenAI, CG-ACT-DEV – the control group with no GenAI access, EG-ALL – Developers who had access to GenAI

blueoptima

2. Licence-based Group Comparison Validation Chi-Square analysis: We applied a Chi-Squared analysis on the Licence-based Group once CAD was used to analyse the commits made by the developers considered in this analysis. This analysis is to establish the extent of GenAI usage as observed from their committed code and the timing of GenAI usage by the treatment group.

   The Chi-Squared test was used to analyse the association between early adoption of AI tools (before official licence assignment) and improvement in performance (measured as Avg. BCE/Day). The test results show a p-value of 0.35, indicating no significant association between early adoption and performance improvement.

3. T-Tests: To further explore the performance differences and usage of GenAI in the CAD-based Group Comparison independent sample t-tests were conducted to compare the productivity and quality between High, Low, and No AI-Contributing Developers for both pre and post timeframes.

Productivity				Quality		
Developer Group	Pre BCE/ Day	Post BCE/ Day	% Change	Pre Aberrancy %	Post Aberrancy %	% Change
High AI-Contrib.	2.01	2.18	+8.4%	5.99	6.01	+0.33%
Low AI-Contrib.	2.59	2.64	+1.93%	5.92	5.91	+0.16%
No AI-Contrib.	1.92	1.88	-2.08%	6.27	6.33	+0.9%

Figure 4: Productivity and quality metrics compared across 3 developer groups: High AI-Contrib. – developers with above average levels of AI-generated code in their commits, Low AI-Contrib. – developers with below average levels of AI-generated code in their commits, No AI-Contrib. - developers not committing any AI-generated code

blueoptima

The percentage changes were calculated as follows:

$$\text{Percentage Change} = \left(\frac{\text{Post}-\text{Pre}}{\text{Pre}}\right) \times 100$$

These statistical analyses enable us to quantify the impact of GenAI on software developer performance and draw conclusions regarding productivity and quality improvements.

Results

The means for the four groups of developers included in the quasi-experimental analysis are provided below.

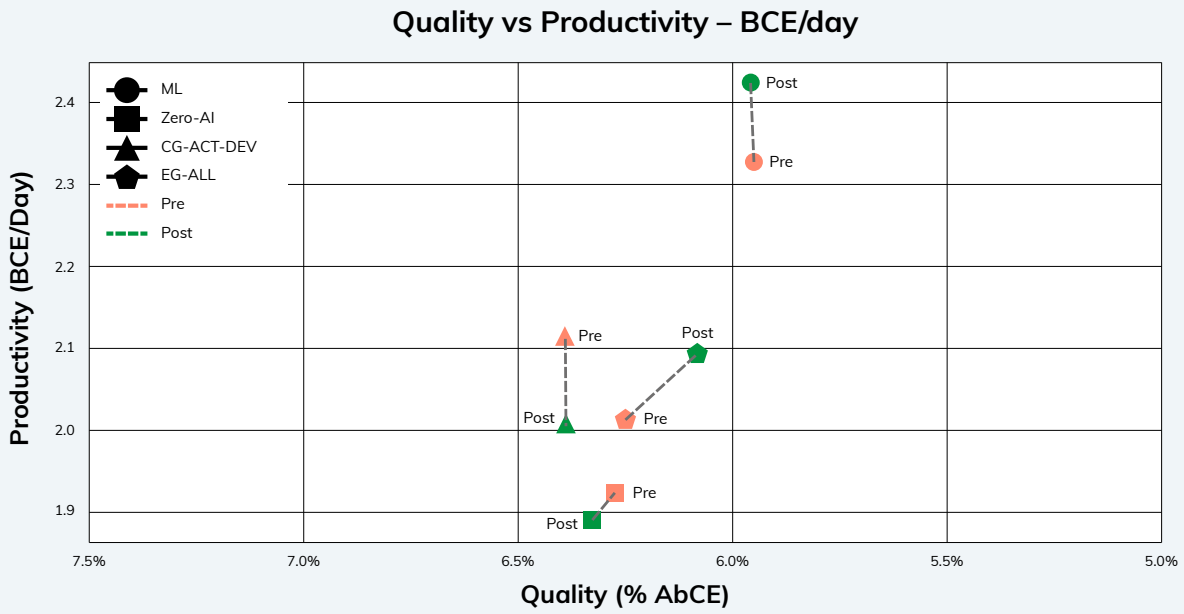


Figure 5: Productivity and quality metric changes compared across 4 developer groups: ML – developers with GenAI access who made changes identified as GenAI, Zero-AI – developers with GenAI access who made changes identified as GenAI, CG-ACT-DEV – the control group with no GenAI access, EG-ALL – Developers who had access to GenAI

blueoptima

The results of the Licence-based and CAD-based quasi-experimental grouping ANOVAs are provided below.

ANOVA Results	Productivity (BCE/Day)	Quality (% Aberrant Coding Effort)
Licence-based Group Comparison ANOVA		
Main Effect of Group	$F = 28.98, p < 0.00001$	$F = 11.97, p < 0.001$
Main Effect of Timeframe	$F = 9.96, p < 0.01$	$F = 4.61, p < 0.05$
Interaction Effect (Group*Timeframe)	$F = 17.87, p < 0.0001$	$F = 1.03, p > 0.31$
CAD-based Group Comparison ANOVA		
Main Effect of Group	$F = 959.14, p < 0.00001$	$F = 31.51, p < 0.00001$
Main Effect of Timeframe	$F = 2.26, p > 0.13$	$F = 4.18, p > 0.45$
Interaction Effect (Group*Timeframe)	$F = 11.44, p < 0.001$	$F = 2.35, p > 0.57$

Figure 6: Anova results across the following comparisons:
1) Licence-based: Main Effect of Group – developers granted access to GenAI compared with those not granted access, Main Effect of Timeframe – the pre and post timeframes, Interaction Effect – the combined comparison
2) CAD-based: Main Effect of Group – developers with AI-generated Code and compared with those who haven't had any AI-generated code, Main Effect of Timeframe – the pre and post timeframes , Interaction Effect – the combined comparison

blueoptima

Licence-based Group Comparison ANOVA

The Licence-based Group Comparison ANOVA aimed to compare the performance of developers officially granted access to GenAI tools (Experimental Group) with those who were not (Control Group).

Productivity (BCE/Day): Developers in the Experimental Group (EG-ALL) showed a notable improvement in productivity, increasing from 2.011 BCE/Day in the pre-GenAI period to 2.092 BCE/Day in the post-GenAI period, a 3.99% increase. In contrast, the Control Group (CG-ACT-DEV) experienced a decrease in productivity from 2.116 BCE/Day to 2.007 BCE/Day, a decline of 5.12%. Within the Experimental Group, the ML Group saw an increase from 2.32 BCE/Day to 2.42 BCE/Day, while the Zero-AI Group observed a decrease from 1.92 BCE/Day to 1.89 BCE/Day.

Quality (% Aberrant Coding Effort): In terms of quality, the Experimental Group exhibited a slight improvement, with a decrease in Aberrant Coding Effort from 6.24% to 6.08%, translating to a 2.63% reduction in aberrancy. The Control Group's quality remained relatively stable, with a minor reduction from 6.393% to 6.390%, equivalent to a 0.05% decrease. Within the Experimental Group, the ML Group's aberrancy increased from 5.95% to 5.96%, whereas the Zero-AI Group saw an increase from 6.28% to 6.33%.

The ANOVA results indicated significant differences in productivity and quality between the Experimental and Control groups. The Main Effect of Group (Experimental vs. Control) on productivity was significant ($F = 28.98$, $p < 0.00001$), and the Main Effect of Timeframe (Pre vs. Post) was also significant ($F = 9.96$, $p < 0.01$). The Interaction Effect between Group and Timeframe was significant as well ($F = 17.87$, $p < 0.0001$), suggesting that the productivity improvements were notably influenced by the use of GenAI tools.

In terms of quality, the Main Effect of Group was significant ($F = 11.97$, $p < 0.001$), while the Main Effect of Timeframe was also significant ($F = 4.61$, $p < 0.05$). The Interaction Effect between group and timeframe was not significant ($F = 1.03$, $p > 0.31$), indicating that the quality changes were relatively stable over time.

CAD-based Group Comparison ANOVA

The CAD analysis ANOVA evaluated the effects of GenAI-authored commits on productivity and quality by comparing developers who used GenAI (ML group) with those who did not (Zero-AI group).

These results independently confirm the positive impact of GenAI on productivity observed in the licence-based comparison while also showing that using GenAI tools does not negatively impact code quality

Productivity (BCE/Day): Developers in the ML group experienced a 4.09% increase in productivity, from 2.32 BCE/Day to 2.42 BCE/Day. In contrast, the Zero-AI group saw a 1.81% decline in productivity, from 1.92 BCE/Day to 1.89 BCE/Day.

Quality (% Aberrant Coding Effort): In terms of quality, the ML group observed a slight drop, with aberrancy increasing from 5.95% to 5.96%, a 0.09% rise. The Zero-AI group also saw a minor degradation in quality, with aberrancy increasing from 6.28% to 6.33%, a 0.89% increase.

The ANOVA results indicated significant differences in productivity between the groups. The Main Effect of Group (ML vs. Zero-AI) on productivity was significant ($F = 959.14$, $p < 0.00001$), and the Interaction Effect between group and timeframe was also significant ($F = 11.44$, $p < 0.001$). For quality, the Main Effect of Group was significant ($F = 31.51$, $p < 0.00001$), but the main effect of the timeframe was not significant ($F = 4.18$, $p > 0.45$). The Interaction Effect between group and timeframe was not significant ($F = 2.35$, $p > 0.57$), indicating that the quality changes were consistent across both groups over time.

These results independently confirm the positive impact of GenAI on productivity observed in the licence-based comparison while also showing that using GenAI tools does not negatively impact code quality.

Licence-based Group Validation Chi-Square Analysis

Extent of GenAI Usage: Of the 3268 developers in the experimental group, 392 (12%) were found to commit ML-authored code. This indicates a notable subset of developers actively using GenAI tools.

Timing of GenAI Usage: Among the 394 developers who committed ML-authored code, 252 (64%) had their first ML-authored commit date well before the official date of Copilot licence assignment. However, the Chi-Squared test indicates that there is no significant association between early adoption and performance improvement, with a p-value of 0.35.

Statistical Validation: The Chi-Squared test was used to analyse the association between early adoption of AI tools (before official licence assignment) and improvement in performance (measured as Avg. BCE/Day).

The test results show a p-value of 0.35, indicating no significant association between early adoption and performance improvement.

Specifically, 122 (48%) early adopters saw an improvement in performance, compared to 130 (52%) who did not. In contrast, 1559 (51%) developers who did not adopt early showed improvement, compared to 1457 (49%) who did not.

T-Tests for MLCAD Groupings

The means of the High, Low, and No-GenAI users saw the following results first shown graphically and then with specific values as a table.

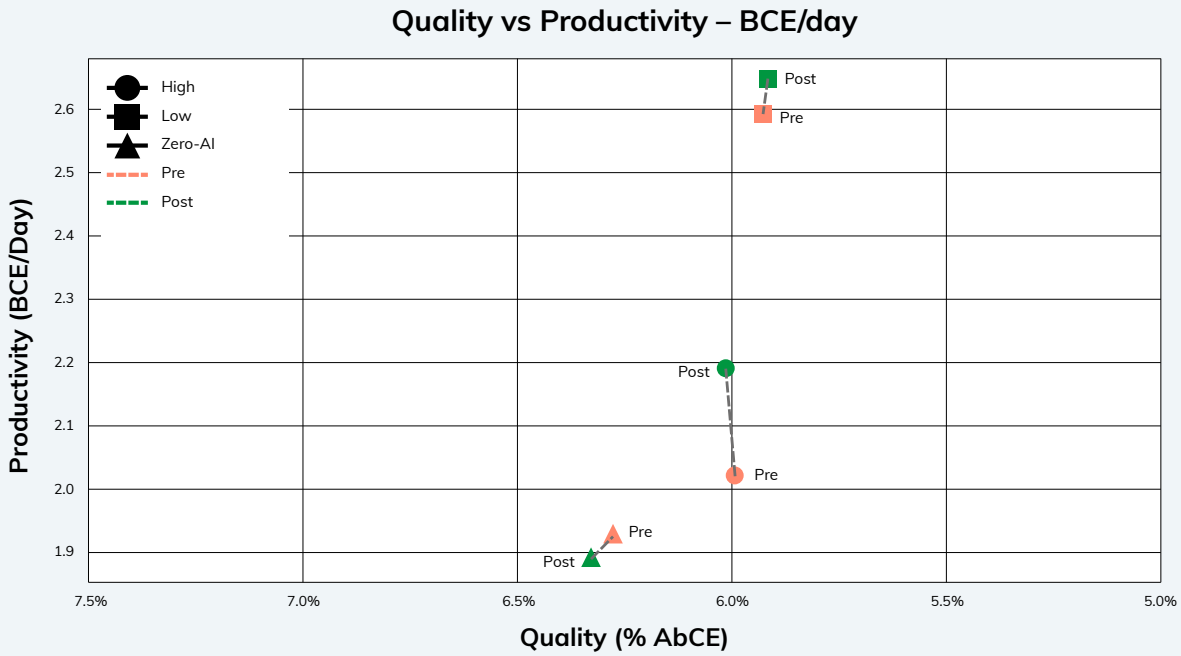


Figure 7: Productivity and quality metrics compared across 3 developer groups: High AI-Contrib. – developers with above average levels of AI-generated code in their commits, Low AI-Contrib. – developers with below average levels of AI-generated code in their commits, No AI-Contrib. – developers not committing any AI-generated code

T-Tests were then used to compare productivity and quality between High, Low, and No AI-Contributing Developers for both pre and post timeframes. The results are summarised in Table 3.

Table 3: T-Test Results for Pre and Post Timeframes

Group Comparison	Pre-Timeframe P-value	Post-Timeframe P-value	Productivity (BCE/Day)	Quality (Aberrancy %)
High vs. Low	7.103e-56	2.268e-36	Significant	Not significant
High vs. None	0.000725	1.766e-21	Significant	Significant
Low vs. None	4.120e-124	2.570e-154	Significant	Significant

Figure 8: T-test comparison on the groups: High – developers with above average levels of AI-generated code in their commits, Low – developers with below average levels of AI–generated code in their commits, None – developers not committing any AI-generated code

blueoptima

The t-test analyses revealed that high and low AI-contributing developers are significantly more productive than non-users in both pre- and post-timeframes. High AI-contributing developers showed higher productivity compared to low AI-contributing developers. In terms of quality, high AI-contributing developers maintained better quality compared to non-users, but there was no significant difference between high and low AI-contributing developers.

Summary of Findings

The results of the Licence-based and CAD-based quasi-experimental grouping ANOVAs indicate that Generative AI (GenAI) positively impacts software developer performance.

In the Licence-based Group Comparison, developers with GenAI access showed a 3.99% increase in productivity, while those without it saw a 5.12% decrease. Quality improved slightly for the Experimental Group with a 2.63% reduction in aberrancy, while the Control Group's quality remained stable.

The CAD-based Group Comparison ANOVA further supported these findings. Developers who used GenAI experienced a 4.09% increase in productivity, while those who did not use GenAI saw a 1.81% decline. In terms of quality, both groups showed minimal changes, with the GenAI users experiencing a slight 0.09% increase in aberrancy and the non-users showing a 0.89% increase.

The significant ANOVA results from both analyses suggest that GenAI tools notably enhance productivity without compromising code quality.

Subsequent T-Tests on the CAD data revealed that High AI-Contributing Developers experienced an 8.4% increase in productivity, and Low AI-Contributing Developers saw a 1.93% increase, while No AI-Contributing Developers saw a 2.08% decline. Quality changes were minor across all groups.

T-Tests confirmed that High and Low AI-Contributing Developers were significantly more productive than non-users, with high contributors maintaining better quality. However, differences in quality between high and low contributors were not significant.

These findings consistently demonstrate the positive effect of GenAI on productivity across different analytical approaches, while also indicating that code quality is maintained or slightly improved with GenAI use.

Discussion

This comprehensive study provides critical insights into the impact of Generative AI (GenAI) tools on software developer performance. It addresses key questions about productivity, code quality, and the complexities of AI integration in software development workflows. Our findings reveal a complex landscape where the benefits of GenAI are significant but vary based on usage patterns and individual developer characteristics.

Widespread Analysis and Limited GenAI Usage

One of the most striking findings of our study is the relatively low incidence of GenAI code being committed without significant rework by developers. In the case of the License-based comparison group, 3,268 were granted access to and officially permitted to use GenAI, and yet only 394 (12%) committed multiple instances of code authored by GenAI that were not materially altered by those developers.

Despite analysing the activity of over 218,000 developers in the GenAI-based comparison group, only 2,062 were found to have material and consistent contributions to version control systems of GenAI-authored code that have not been significantly edited and altered by the developer contributing it. This represents less than 1% of the developer population analysed.

It may be the case that the enterprises for which a number of these 218,000 developers work may prohibit the use of LLM due to information security concerns (Zhu et al., 2024). Whatever the permissiveness of the enterprises considered in terms of the use of GenAI it is clear that the integration of GenAI tools into software development practices is still in its early stages and that the LLMs producing the code have not achieved a level of proficiency in accommodating the context of the target codebase nor understanding the desired requirements to not require further manipulation by developers.

This raises important questions about the nature of the efficiency that is being delivered by GenAI. Specifically, it highlights the fact that while GenAI can compose source code, currently, this source code requires significant alteration before it is appropriately assimilated into a wider codebase to be committed to version control systems. This suggests that if wider use of GenAI is in fact happening in enterprises the use cases must primarily relate to scenarios other than largely autonomous composition of source code capable of delivering value to end users in Production environments.

One of the most striking findings of our study is the relatively low incidence of GenAI code being committed without significant rework by developers

Impact on Productivity and Quality: Licence-based vs. CAD-based Analysis

Our study employed two complementary approaches to assess the impact of GenAI: a licence-based group comparison and a Code Author Detection (CAD) based analysis.

The licence-based quasi-experimental ANOVA revealed a clear positive impact of GenAI tools on developer productivity. The Experimental Group (EG-ALL) showed a 3.99% increase in productivity (BCE/Day) after GenAI implementation, contrasting with the Control Group's (CG-ACT-DEV) 5.12% decrease. Importantly, this productivity gain did not come at the expense of code quality; the Experimental Group saw a 2.63% reduction in Aberrant Coding Effort, indicating a slight improvement in code maintainability.

The CAD-based ANOVA provided further support for these findings. In this analysis, developers who used GenAI (regardless of the level of usage) experienced a 4.09% increase in productivity, while those who did not use GenAI saw a 1.81% decline. This analysis reinforces the positive impact of GenAI on productivity observed in the licence-based comparison. Interestingly, the quality metrics showed minimal increases in Aberrancy across both groups in the CAD-based analysis, with Zero-AI users having a higher increase in Aberrancy (0.89%) as compared to developers using GenAI (0.09%) (reduction in quality).

So, the average increase in productivity across these two approaches is just over 4%. While this may not seem a radical improvement in productivity it should be understood that genuine and enduring improvements in productivity are harder to achieve than one might expect given the volatile and unreliable percentage improvements in measures of productivity that are quoted based on workflow based metrics (e.g. DORA Lead Time to Change, (BlueOptima, 2024)). Such measures are often short-lived as they relate to changing developer behaviour regarding conventions of ways of describing, coordinating, and managing work tasks within and across teams through task tracking systems. Code artefact measures of productivity offer a much deeper and reliable account of productivity. Based on the BlueOptima Global Benchmark of Productivity, Quality, and Cost, an enterprise of 250 software developers, a 4% increase in productivity while maintaining code quality represents a cost efficiency of over two million dollars per year (see Appendix B).

These findings from both ANOVAs challenge the notion that increased productivity through GenAI use necessarily leads to lower code quality. Instead, they suggest that GenAI tools can enhance productivity without negatively impacting code quality.

Adoption Patterns and Performance

The CAD analysis revealed intriguing patterns in GenAI adoption and usage. In the licence-based group, only 12% of developers in the Experimental Group were found to commit materially unaltered ML-authored code, suggesting that the majority either use GenAI tools for line completion, documentation, or other ways not detectable by our current methods or primarily as assistive tools for code ideation and drafting rather than direct code generation. The prevalence of this across the much larger CAD-based sample of more than 218,000 was less than 1%, though the availability of GenAI tools to this entire population is unknown.

The ideal use case for GenAI would be to compose performant, accurate, and appropriate high-quality code that developers can confidently commit to version control systems. This would maximise GenAI's performance impact. However, this study demonstrates that this is not GenAI's predominant use case in practice.

Another important finding was that 64% of developers who committed ML-authored code did so before their official licence assignment date. However, the Chi-Squared test indicates that there is no significant association between early adoption and performance improvement, with a p-value of 0.35.

These results raise important questions about the relationship between developer initiative, tool adoption, and performance enhancement. It suggests that the organisations involved in the study did not have the control they thought they had over the use of software development tooling by their developer populations. It also suggests that those developers more likely to adopt new tooling of their own volition are more likely to be able to yield productivity gains from new tooling.

Levels of AI Integration and Developer Performance

Our analysis of High, Low, and No AI-Contributing Developers across a larger population provided further detail to our understanding of GenAI's impact. High AI-Contributors saw a 8.4% productivity increase, while Low AI-Contributors experienced a 1.93% increase. Conversely, No AI-Contributors faced a 2.08% productivity decline. These findings suggest a positive correlation between AI tool usage and productivity gains, but also hint at potential diminishing returns, given developers of higher attainment seeing less benefit or challenges with very high levels of AI integration, and given the apparent tendency to lower quality of this code.

Interestingly, and perhaps most significantly, Low AI-Contributing Developers emerged as the highest overall performers, balancing high productivity with the best code quality. This suggests an optimal "sweet spot" in AI tool usage where developers leverage AI assistance to enhance their work without over-relying on it. It underscores the importance of human oversight and judgement in effective AI integration.

Finally, researchers (Dell'Acqua et al., 2023) have observed a "levelling" effect of Generative AI outside of the software development industry whereby less capable operatives in a given work task see the largest improvements in performance. This phenomenon is observed in this study too with the High AI-Contributors showing the most liberal use of the technology while also showing the largest improvement in productivity.along with only a very minor decline in quality.

An alternative account could be simpler and that the higher performing developers were less likely to accept code suggestions made by GenAI without ensuring that the code is

elegantly integrated into the wider codebase. This means that they are spending more time editing code and therefore they were seeing less of a productivity improvement. This might also explain why the positive trend in quality for the higher performing developers in contrast to the High AI-Contributors who saw a negative quality trend.

Limitations and Future Research Directions

While our study provides valuable insights, it also has limitations that point to areas for future research:

1. Detection Limitations: Our current methods may not capture all forms of GenAI usage, particularly if developers heavily modify AI-generated code. Future studies could employ more granular tracking methods to capture the full spectrum of AI assistance in development workflows. This would require far more detailed telemetry and reporting from LLM providers regarding usage of their products than is currently available. Once this is understood, the industry can begin to understand if these more limited assistive use cases warrant the significant additional cost of GenAI over and above what is already available in most modern IDEs.
2. Low Code Acceptance Rates: Further research is needed to understand the reasons behind the low acceptance rates of GenAI-authored code. The use case that would maximise GenAI's benefit is that it would compose performant, accurate, and appropriate high-quality code that developers can confidently commit to version control systems. This would maximise GenAI's performance impact. This study shows that this is not what is observed in practice. We need to understand why.
3. Long-term Impacts: This study provides a narrow time-range of GenAI's impact, but longitudinal studies are needed to understand how these effects evolve over time and how they influence career trajectories and skill development.
4. Causality and Self-Selection: The strong performance of early adopters raises questions about the directionality of the relationship between AI tool usage and developer performance. Future research could employ more sophisticated methods to disentangle the effects of tool adoption from pre-existing developer characteristics.

The future of software development likely lies not in a choice between human and AI, but in finding the optimal synergy between developer expertise and AI assistance

5. Contextual Factors: Further research is needed to understand how the impact of GenAI tools varies across different types of development tasks, project contexts, technologies, languages, tenures, seniority levels, roles, etc.
6. Optimal Usage Patterns: The superior performance of Low AI-Contributors warrants deeper investigation into what constitutes optimal AI tool usage and how to achieve it consistently across development teams

Conclusion

This study provides strong evidence that GenAI tools can significantly enhance software developer productivity while maintaining or even improving code quality. However, the impact is not uniform and depends heavily on how these tools are integrated into development workflows. The most effective approach appears balanced, where AI augments rather than replaces human expertise.

The surprisingly low adoption rate of GenAI tools among developers highlights both the potential for significant future impact as adoption increases and the need for a better understanding of adoption barriers. As GenAI tools evolve and become more prevalent in software development, ongoing research will be crucial to understand their full impact and develop best practices for their integration. The future of software development likely lies not in a choice between human and AI, but in finding the optimal synergy between developer expertise and AI assistance, and in effectively scaling the adoption of these powerful tools across the developer community.

Recommendations for Software Development Executives

Based on the study's findings, software development executives should consider the following when implementing GenAI in an enterprise setting:

1. Address Low Adoption Rates	Given the extremely low percentage of developers committing unaltered GenAI-authored code, investigate specific barriers to effective use. Consider surveying developers to understand the challenges in integrating AI-generated code into existing codebases.
2. Balanced Integration Approach	Develop guidelines that encourage GenAI use while emphasising the importance of human oversight. The study suggests an optimal "sweet spot" where AI augments rather than replaces human expertise.
3. Targeted Support	Given the varied impacts across developer groups, consider tailored approaches for different skill levels and roles. Focus on helping developers effectively leverage AI tools within their existing workflows.
4. Continuous Evaluation	Implement ongoing monitoring of both productivity and quality metrics. This study used BCE/Day and Aberrant Coding Effort; consider adopting these or developing similar metrics suitable for your organisation. Monitoring ensures that improvements in one area do not come at the expense of another, maintaining a holistic view of performance.
5. Early Adoption Insights	The strong performance of early adopters suggests potential benefits in identifying and learning from developers who quickly adapt to new tools. Consider mechanisms to share best practices across teams.
6. Address Potential Skill Gaps	Given the performance decline observed in non-AI users, develop strategies to ensure all developers can maintain competitiveness as AI tools become more prevalent.
7. Optimise Usage Levels	Given that Low AI-Contributing Developers showed the best overall performance, investigate what constitutes optimal AI tool usage in your specific context.
8. Long-term Impact Assessment	As this study covered a limited timeframe, plan for long-term evaluation of GenAI's impact on productivity, code quality, and developer skill evolution.

These recommendations focus on pragmatic steps based directly on this study's findings, aiming to maximise the benefits observed while addressing potential challenges in GenAI adoption and use.

References

BlueOptima. (2024). DORA lead time to change – useful but inadequate. Retrieved from https://www.blueoptima.com/resource/dora-lead-time-to-change-useful-but-inadequate/

Almeida Y, Albuquerque, Filho, Muniz, de Farias Santos, Perkusich, Almeida H, Perkusich A (2024) AICodeReview: Advancing code quality with AI-enhanced reviews https://www.sciencedirect.com/science/article/pii/S2352711024000487

Chatterjee, S., Liu, C. L., Rowland, G., & Hogarth, T. (2024). The impact of AI tool on engineering at ANZ Bank: An empirical study on GitHub Copilot within corporate environment. arXiv. https://doi.org/10.48550/arXiv.2402.05636

Peng, Kalliamvakou, Cihon, Demirer (2023) The Impact of AI on Developer Productivity: Evidence from GitHub Copilot. arXiv. https://arxiv.org/pdf/2302.06590

Lwakatare, L. E., Raj, A., Bosch, J., Olsson, H. H., & Crnkovic, I. (2022). A taxonomy of software engineering challenges for machine learning systems: An empirical investigation. In Agile Processes in Software Engineering and Extreme Programming (pp. 227–235). Springer, Cham. https://doi.org/10.1007/978-3-030-98103-0_18

Rahman, M. M., Parnin, C., & Williams, L. (2023). Machine Learning for Software Engineering: A Systematic Mapping. ACM Computing Surveys, 55(5), 1–36. https://doi.org/10.1145/3526064

Sobreira, V., Durelli, V. H., Peixoto, M., Hui, J., & Treude, C. (2023). Investigating the Use of ChatGPT for Code Documentation Generation. In 2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP) (pp. 323-332). IEEE. https://doi.org/10.1109/ICSE-SEIP58684.2023.00044

The Economist. (2024, July 2). What happened to the artificial-intelligence revolution? The Economist. Retrieved from https://www.economist.com/finance-and-economics/2024/07/02/what-happened-to-the-artificial-intelligence-revolution

Wessel, M., Steinmacher, I., Wiese, I., & Gerosa, M. A. (2022). Should I strive to be a top developer? Tension and satisfaction among top developers in open source projects. IEEE Transactions on Software Engineering, 48(7), 2650–2665. https://doi.org/10.1109/TSE.2021.3053830

Xu, X., Wang, Y., & Zhang, T. (2022). Exploring the Impact of Artificial Intelligence on Software Development: Opportunities and Challenges. In 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER) (pp. 612–616). IEEE. https://doi.org/10.1109/SANER53432.2022.00071

Zhu, B., Mu, N., Jiao, J., & Wagner, D. (2024). Generative AI security: Challenges and countermeasures. arXiv. https://doi.org/10.48550/arXiv.2402.12617v1

Appendices

Appendix A

Ways in which initially GenAI authored code becomes human authored.

BECOMES HUMAN-AUTHORED (I.E. IMPACTS STYLE & STRUCTURE)	REMAINS AI-AUTHORED (I.E. DOES NOT IMPACT STYLE & STRUCTURE)
Changes in control flow: Modifying loops, conditionals, branching statements	Simple comments, variable renaming
Adding or removing function calls: Introducing new functionality or removing unnecessary code	Changing variable names within functions
Altering data structures: Switching array/list implementations, modifying object hierarchy	Formatting changes, whitespace adjustments
Introducing custom logic: Implementing unique algorithms or data processing steps	Adding documentation comments
Changing variable types: Modifying data types of variables and function arguments	Changing variable names without type changes
Combining or splitting code blocks: Merging or dividing functions, conditional statements	Reordering lines of code within a block

Appendix B

Calculations of the impact of a 4% increase in productivity that maintains quality based on the BlueOptima Global Benchmark.

VARIABLE	VALUE	COMMENTS
Number of developers	250	
BGB Universe Avg. BCE/Day	1.9	
BGB Universe Avg. Cost/Day	432	
BGB Universe Avg. Cost/BCE	$227	
Number of working days in a calendar	255	
Total BCE	121125	(BGB Universe Avg. BCE/Day * Number of working days in a calendar * Number of developers)
4% increase in Avg. BCE/Day	1.976	
Improved Cost/Day after 4% increase in Avg. BCE/Day	$219	
Capacity cost saving	$2,012,538	(Total BCE * (BGB Universe Avg. Cost/Day – ((BGB Universe Avg. Cost/Day * BGB Universe Avg. BCE/Day) / 4% increase in Avg. BCE/Day))

p +44 207 100 8740
e enquiries@blueoptima.com
w www.blueoptima.com