Assessing the Use of a Code Generation Assistant in Professional Software Development: An Experience Report

Luiz Fernando Mendes Osorio
Universidade Federal do Piauí
Teresina, Piauí, Brasil
fernando.mendes@ifpi.edu.br

Pedro de A. dos Santos Neto
Universidade Federal do Piauí
Teresina, Piauí, Brasil
pasn@ufpi.edu.br

Guilherme Avelino
Universidade Federal do Piauí
Teresina, Piauí, Brasil
gaa@ufpi.edu.br

Werney Ayala Luz Lira
Instituto Federal do Piauí
Teresina, Piauí, Brasil
werney@ifpi.edu.br

ABSTRACT

AI-assisted code generation tools, such as GitHub Copilot, have gained attention from the software industry due to their potential to support development tasks. This paper presents an experience report from a case study on the adoption of GitHub Copilot by a team of developers at the Superintendence of Information Technology of the Federal University of Piauí (STI/UFPI), aiming to evaluate the effects of using Copilot in a real-world environment involving legacy system maintenance. The study was carried out over three months, comparing periods with and without the use of the tool. Key metrics such as code change rate and developers' perceptions were analyzed. The results indicate a increase in the code change rate for most developers, particularly in bug-fixing tasks, while customization tasks showed a more nuanced outcome. Participants also reported positive feedback regarding Copilot's usefulness. This paper discusses the challenges faced during the study's execution, methodological limitations, and proposes directions for future research on the impact of AI assistants in diverse software development contexts.

KEYWORDS

GitHub Copilot, Code Generation Tools, Empirical Study, Legacy Systems, Artificial Intelligence, Experience Report

1 Introduction

The increasing complexity of software systems and the demand for agility in software development have led engineering teams to adopt new tools that accelerate the construction of high-quality solutions. In this context, code generation tools based on Artificial Intelligence (AI), such as GitHub Copilot, have emerged as prominent tools for accelerating software development tasks. These tools rely on Large Language Models (LLMs), such as Codex, which are capable of suggesting complete code snippets based on comments or previously written fragments by the programmer [8, 11, 30].

Although the literature has begun to explore the potential of such tools in academic and simulated environments, there is still a limited number of studies that evaluate their effectiveness in real-world settings, where professional teams work with production systems [20, 21, 26, 33]. For instance, the study by [19] introduced NoCodeGPT, a customized interface designed to enable the creation of small web applications without writing code. Their findings highlight that general-purpose chat interfaces, such as ChatGPT, are not well-suited for novice developers, reinforcing the need for contextualized and domain-specific adaptations of LLM-based tools.

Given the diversity and complexity of legacy systems, integrating new tools into the workflow of experienced developers demands a more careful and contextualized evaluation. Unlike controlled or academic environments, professional software projects often involve heterogeneous technologies, strict delivery deadlines, and organizational constraints that directly affect how such tools are perceived and adopted. Therefore, assessing AI-assisted code generation in real-world settings is essential to understanding its actual impact on productivity, code quality, and developer experience—beyond the promising results observed in experimental or educational contexts [24].

This paper presents an experience report from a case study on the adoption of GitHub Copilot by a team of developers at the Superintendence of Information Technology of the Federal University of Piauí (STI/UFPI). The objective was to observe, in a real work environment, the potential impacts of the tool on the performance and perceived quality of professionals involved in the development of institutional systems.

The case study was conducted over a period of three months (March, April, and May 2025), including the collection of task execution time data and qualitative analysis of participants' perceptions. The results are discussed from the perspective of perceived benefits, challenges encountered, and lessons learned from using the tool in a corporate development setting.

The main research question guiding this report is:

> **How does the use of GitHub Copilot impact the performance and perception of professional developers in a real-world software development environment?**

The main contributions of this paper can be summarized as follows:

- **For industry:** We provide empirical evidence on the use of Copilot in a realistic scenario involving legacy systems in the public sector, a context that remains underexplored in the literature.

- **For academia:** We present a case study employing mixed methods, whose dataset and methodology may support future research. Furthermore, we connect the practical challenges faced in a real-world industrial context to formal threats to validity, offering transferable lessons for similar empirical studies.

The remainder of this paper is organized as follows: Section 2 discusses related work that investigated the use of Copilot and similar tools in industrial contexts. Section 3 presents the theoretical background on AI-assisted code generation tools and previous studies on GitHub Copilot. Section 4 describes the methodological approach adopted in the case study. Next, Section 5 presents the results obtained from the quantitative and qualitative data analysis. Section 6 presents the lessons learned. Section 7 details the threats to the validity of the findings. Finally, Section 8 summarizes the study's contributions and proposes directions for future work.

2 Related Work

Several recent studies have investigated the impact of AI-assisted code generation tools in professional contexts. In particular, case studies have been conducted in companies to evaluate the effectiveness of GitHub Copilot and similar tools in real-world development environments.

Pandey et al. [25] conducted a study at Cisco Systems to assess the adoption of GitHub Copilot in large-scale projects. The authors reported gains of up to 50% in tasks such as documentation and unit testing, and approximately 30–40% in debugging and repetitive tasks. Despite the observed benefits, limitations were identified when working with languages such as C/C++ and in contexts involving multiple interdependent files.

Bakal et al. [1] reported on the large-scale use of Copilot at ZoomInfo, involving approximately 400 developers. The study found an average suggestion acceptance rate of 33% and an estimated time savings of 20%, along with generally positive perceptions among the developers.

Cavalcante et al. [6] analyzed the application of GitHub Copilot, ChatGPT, and Amazon CodeWhisperer in a research and development organization with 57 participants. The study highlighted improvements in productivity, creativity, and prototyping speed, while also warning of potential risks related to excessive reliance on such tools.

Raghavan [28] presented an industrial case study using the SPACE framework to measure productivity and quality in a corporate environment. GitHub Copilot was found to have a positive impact, particularly in initial coding tasks and in supporting rapid solution generation.

Chatterjee et al. [7] described the experience of over one thousand engineers at ANZ Bank using Copilot. The authors adopted a quasi-experimental design with control and treatment groups to demonstrate gains in productivity and code quality, with particular attention to security and compliance issues—crucial aspects in the financial sector.

In comparison to these studies, the present work contributes by examining a public-sector setting focused on the development and maintenance of legacy systems. It emphasizes tasks related to evolutionary and corrective maintenance and adopts a mixed-methods approach that combines quantitative and qualitative data to provide an integrated perspective on the tool's impact on both technical performance and developer perception.

3 Theoretical Background

This section presents the evolution of code generation tools, the main indicators used for their evaluation, and the foundations of empirical studies in Software Engineering, with a particular focus on GitHub Copilot—the central theme of this work.

3.1 Evolution of Code Generation Tools

Since the early days of programming, various tools have been developed to automate or support the coding process. Initial efforts date back to utilities such as Lex and Yacc in the 1970s, which enabled the automatic generation of parsers and syntax analyzers based on formal specifications [15]. These tools marked the beginning of a path toward automation focused on reducing repetitive tasks and increasing the reliability of the generated code.

Over time, the concept of code generation evolved. Modern IDEs have incorporated features such as autocomplete, automated refactoring, and semantic navigation. Tools like IntelliSense [17] and capabilities available in Visual Studio Code [18] have enhanced productivity and fluency in the development process by offering contextual suggestions based on code syntax and semantics. Studies such as [13] and [27] highlight the role of these technologies in improving code quality and development efficiency.

3.2 GitHub Copilot and AI-Based Code Generation

GitHub Copilot is an AI-assisted code generation tool developed by OpenAI in partnership with GitHub [11]. The technology behind Copilot is based on Codex [8], a language model derived from GPT-3 [5], specialized in programming tasks. More recently, Copilot began using the GPT-4 model, expanding its contextual understanding and ability to generate functional code [12].

Copilot operates as an extension integrated into the code editor, suggesting lines, blocks, or even complete functions based on content previously written by the developer. Unlike traditional autocomplete systems, the tool is capable of interpreting the purpose of the code being written, taking into account comments, variable names, and file history. This makes its suggestions more relevant and potentially productive, as noted by [26].

Recent studies have evaluated Copilot's effectiveness in different contexts. [21] show that the tool can reduce coding time for routine tasks, although it presents limitations in situations requiring deep understanding of business rules or the architecture of legacy systems. Other authors highlight the risk of incorrect or insecure suggestions, reinforcing the need for human oversight [22, 34]. Nonetheless, there is consistent evidence that Copilot can enhance productivity, especially in repetitive, low-risk tasks that involve recognizable syntactic patterns. Previous work by the authors has also explored the tool's use in educational settings, showing that Copilot can reduce development time [16, 24].

3.3 Evaluating Code Generation Tools

Evaluating the impact of code generation tools requires indicators that measure not only efficiency but also code quality and usability. The literature highlights several quantitative and qualitative metrics applied in different contexts [3, 9].

Among the most commonly used indicators are:

- **Task Completion Time**: used to measure productivity gains in specific tasks [26];
- **Code Correctness**: assesses whether the generated code correctly fulfills task requirements [34];
- **Validity and Compilability**: checks whether the generated code is syntactically valid [9];
- **Security**: analyzes common vulnerabilities introduced automatically [22];
- **Maintainability and Cyclomatic Complexity**: evaluates the long-term impact of the generated code [9, 21];
- **User Feedback**: gathers developer perceptions about the tool's usefulness [10, 27];
- **Suggestion Acceptance Rate**: quantifies how many of the tool's suggestions were actually adopted [14, 21].

In the context of this case study, two of these indicators were applied: **task completion time**, which allowed the measurement of the effort required to complete tasks with and without GitHub Copilot; and **user feedback**, obtained through a structured qualitative questionnaire. Other relevant indicators, such as security, code correctness, and suggestion acceptance rate, were not applied due to limitations in scope, tools, or feasibility in a production environment.

3.4 Empirical Studies in Software Engineering

Empirical studies play a central role in Software Engineering because they make it possible to validate, in real-world settings, the impact of tools and practices on productivity and development quality [29, 32, 35]. These studies can take different forms—controlled experiments, case studies, and surveys—with case studies being the most appropriate when the goal is to observe phenomena in natural, uncontrolled environments.

The use of empirical studies to assess GitHub Copilot has proven to be strategic for understanding its practical impact. According to [32], this approach makes it possible to capture not only objective results but also subjective aspects of the developer experience, such as confidence, satisfaction, and fluency in tool usage. By conducting a case study in a corporate environment, as presented in this paper, it is possible to more accurately identify the factors that contribute (or not) to the successful adoption of AI tools in daily software development routines.

4 Methodology

This article adopts an empirical approach, using a case study design to investigate the impact of GitHub Copilot in a real-world software development environment. The study was conducted with the development team at the Superintendence of Information Technology (STI) of the Federal University of Piauí (UFPI), responsible for maintaining and evolving the university's institutional systems.

The methodology is based on the *Technical Action Research* (TAR) model [31], which combines elements of Action Research with principles from Design Science. This approach is appropriate for research involving emerging technologies, as it allows the investigation of real-world phenomena while seeking to solve practical problems and produce scientific knowledge. In this study's context, TAR enabled the observation of the tool's usage in a professional setting without artificial interventions, respecting the developers' natural workflow.

The research is applied in nature, with a mixed-methods approach involving the collection and analysis of both quantitative data (related to execution time and code changes) and qualitative data (developer perceptions). The study is exploratory and descriptive in design, aiming to understand the effects of the tool on the team's day-to-day development activities.

Figure 1 illustrates the methodological workflow adopted, divided into two main phases: **Data Collection** and **Analysis**. The data collection phase included initial participant training, instrumentation for automatic task tracking, and the administration of a perception questionnaire. The data were stored in `.csv` files and subsequently preprocessed. The analysis phase involved data consolidation, statistical testing, and qualitative analysis based on categories derived from coding. The ultimate goal was to answer the central research question of the study.

4.1 Data Collection

The data collection process began with the **Initial Training** phase. This preparatory stage involved the installation and configuration of the GitHub Copilot tool within the participants' development environments. To enable its use in Eclipse, the Copilot4Eclipse extension (v1.3.0) was installed, which is compatible with version 2023-06 of the IDE.

Three developers from STI participated in the study, hereafter identified as Dev 1, Dev 2, and Dev 3 to ensure anonymity. Their professional experience in software development totals 13, 15, and 14 years, respectively. A significant portion of this time has been dedicated to the legacy systems that are the focus of this study, with each developer having extensive familiarity—over 8 (Dev 1), 7 (Dev 2), and 8 (Dev 3) years—working directly with these systems.Their roles involve both corrective and evolutionary maintenance of UFPI's corporate systems.

The data collection for this study spanned two distinct three-month periods to allow for a comparative analysis. The baseline period, representing the work **without Copilot**, corresponds to tasks performed in *March, April, and May of 2024*. The intervention period, **with Copilot**, covered the same months one year later: *March, April, and May of 2025*, after the tool was installed and ready to use. During this three-month intervention period, all three participating developers consistently used the GitHub Copilot extension in their daily development workflow. Adherence to this protocol was confirmed through periodic monitoring by the first author. The detailed distribution of tasks collected and filtered from these periods is presented in the Results section (Table 1).

Data collection relied on two main sources: the task management tool used by the team and the Git repositories linked to the systems. Execution time was measured from the moment a task was marked as *"In Progress"* until its transition to a completion status, such as *"To be Validated"*, *"Under Validation"*, or *"Completed"*. It is important

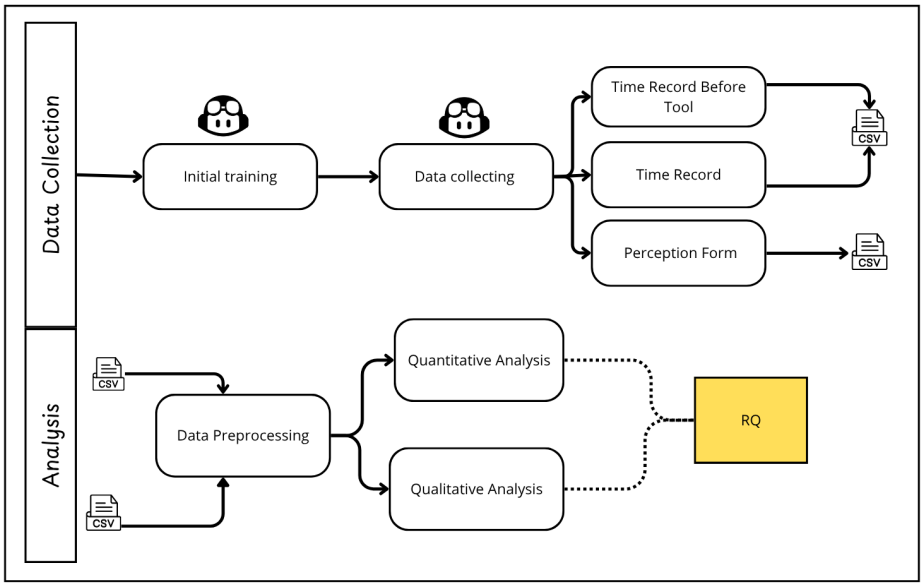

Figure 1: Methodological workflow.

to note that, within the team's workflow, a task only transitions to these completion statuses after the corresponding code changes have been validated against the requirements and successfully deployed to the production environment. This process ensures that all tasks included in the study's dataset represent functional and delivered work. The volume of source code changes was obtained through the commits associated with each task.

Figure 2 shows an example of the task management interface used by the STI/UFPI team. The image highlights three key elements, marked by numbered red arrows:

(1) In the upper left corner, the task type (*Sustaining - Bug*) and its unique reference number in the system (#69645) are displayed, which allows for tracking and categorization of internal demands.
(2) On the right, the task status is marked as *Completed*.
(3) In the bottom right section, the *Associated Revisions* panel links the task to commits in the Git repository. This association is essential for establishing traceability between code changes and the recorded demand, enabling effort analysis.

Ethical Considerations. All participants were informed of the study's objectives and took part voluntarily. Data were anonymized and treated confidentially, in accordance with ethical guidelines for scientific research. Participants signed an Informed Consent Form (ICF).

4.2 Data Analysis

The purpose of the data analysis is to investigate how the use of GitHub Copilot affected developer performance in a professional setting. The analysis approach combined both quantitative and qualitative techniques.

Preprocessing: Before analysis, the collected data were reviewed to ensure consistency and to remove any anomalies. Tasks with incomplete information or without proper links to commits were excluded.

Quantitative Analysis: The quantitative analysis focused on task execution time before and after the introduction of Copilot. Data were grouped by developer and task type, enabling comparisons between the periods and allowing the assessment of potential efficiency gains.

Qualitative Analysis: In parallel, developers' perceptions regarding the use of the tool were collected. Responses were analyzed based on thematic categories, allowing the identification of positive aspects, perceived limitations, and suggestions for improvement. The analysis followed the Content Analysis approach proposed by Bardin [2], aiming to highlight recurring patterns in the reported experiences.

The adopted methodology allows for the observation of the effects of GitHub Copilot in a real development environment, respecting the team's natural task flow. The analyses based on the collected data support the discussions presented in the following sections of this paper.

5 Results

This section presents the results obtained from the analysis of data collected from the developers of the Superintendence of Information Technology (STI) at the Federal University of Piauí (UFPI), during periods with and without the use of GitHub Copilot. The analysis

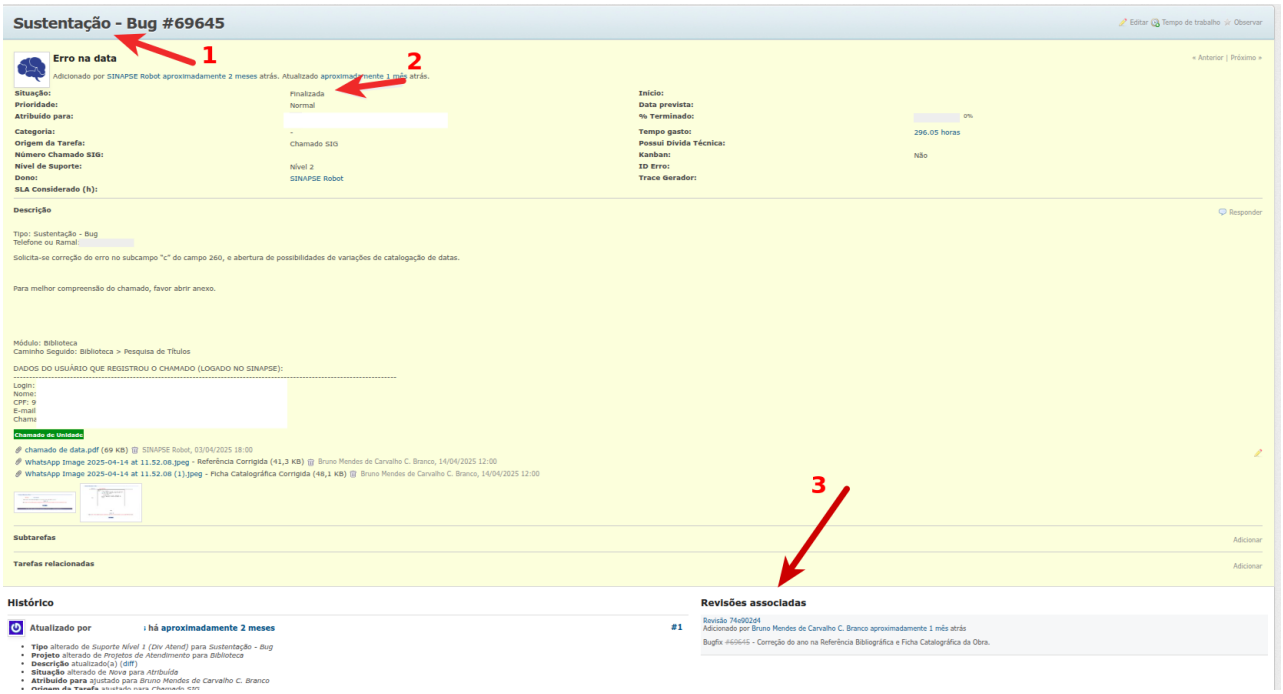

Figure 2: Example of a task in the STI task management system, highlighting (1) task number and type, (2) status and recorded time, and (3) association with Git commits.

includes metrics related to the rate of code changes and developers' perceptions of the tool.

Before conducting the analysis, a data preprocessing step was performed to ensure consistency and reliability. Initially, the tasks exported from the task management system were cross-referenced with commit records from the Git repository to ensure a valid link between the recorded activities and the corresponding source code changes. Tasks without a clear association to Git revisions were discarded. Tasks with incomplete information—such as missing start or end dates, or lack of code change data—were also excluded. In cases where a task had multiple revisions (commits), the data were consolidated per task by summing the total lines of code changed and preserving the original start and end dates.

After this refinement, a clean dataset was obtained for quantitative analysis, structured in CSV files, as documented in the project repository [23], specifically in the files `estudo-de-caso-qualitativo.csv` and `estudo-de-caso-quantitativo.csv`.

To better characterize the dataset analyzed in this study, Table 1 presents the distribution of tasks by developer, categorized by type and period (with or without GitHub Copilot). Developer 1 accounted for the largest volume of tasks, totaling 63 relevant activities: 38 classified as *BugFix* and 25 as *Customization*. Developer 2 contributed with 19 tasks (5 *BugFix* and 14 *Customization*), while Developer 3 performed 10 tasks (7 *BugFix* and 3 *Customization*).

This refined subset of 92 tasks serves as the foundation for the quantitative analysis described in the following subsections.

The primary metric used in this study was the **rate of code changes**, calculated as the ratio between the total lines of code changed (additions + deletions) and the time spent to complete the task.

Table 1: Number of tasks by developer, type, and period

DevID	Type	No Copilot	Copilot	Total
1	BugFix	21	17	38
1	Customization	15	10	25
2	BugFix	5	0	5
2	Customization	9	5	14
3	BugFix	4	3	7
3	Customization	1	2	3
Total		**55**	**37**	**92**

It is crucial to clarify that this metric is not a direct measure of productivity or code quality. Rather, it was chosen as a proxy to quantify and compare the **development effort** between the two periods. In this context, a higher rate (more LOC changed per hour) is interpreted as a *reduction in the time-based effort* required to implement the necessary code for a given task.

We acknowledge that, especially with tools like Copilot, some of these code changes might stem from automated refactoring not strictly related to the task's core requirements. This is a known limitation of using LOC-based metrics in this context and is further discussed as a threat to construct validity in Section 7. The equation is defined as follows:

$$\text{Rate of Code Changes (LOC/h)} = \frac{\text{Lines of Code Changed}}{\text{Time Spent (hours)}} \quad (1)$$

This metric was uniformly applied in analyses by developer, by team, and by task type, enabling comparisons across tasks of different durations and scopes.

The analysis of results is divided into two complementary parts. First, the **quantitative analysis** focuses on the rate of code changes from different perspectives: by developer (Section 5.1), aggregated by team (Section 5.2), and by task type (Section 5.3). Then, the developers' perceptions of GitHub Copilot are discussed, based on **qualitative** data obtained through a questionnaire conducted at the end of the experiment, in Section 5.5. This integrated approach provides a broader understanding of the tool's effects in the professional context under investigation.

5.1 Rate of Code Changes by Developer

Table 2 presents the average (Mean) and median code change rates (LOC/h) for each developer, where 'n' indicates the number of tasks performed per period. The data show that all three developers experienced an increase in their average and median rates during the period with Copilot.

Table 2: Code change rate (LOC/h) by developer and period

ID	Period	Mean	Median	n
1	Without Copilot	14.38	9.83	36
1	With Copilot	27.45	22.65	27
2	Without Copilot	5.01	4.12	14
2	With Copilot	15.06	15.06	5
3	Without Copilot	12.69	8.89	5
3	With Copilot	24.99	20.91	5

Analyzing the individual performance, Developer 1's average rate increased from 14.38 to 27.45 LOC/h. Developer 2 showed the largest relative growth, with the average rate rising from 5.01 to 15.06 LOC/h. Developer 3's rate also grew, from 12.69 to 24.99 LOC/h. The median values followed this upward trend for all developers, suggesting the performance gains were consistent and not merely driven by outlier tasks.

To assess the statistical significance of the observed differences, the non-parametric Mann-Whitney U test was applied, an appropriate choice for the asymmetric distributions of the data. To account for multiple comparisons across the three developers, a Bonferroni correction was applied, setting the significance level at $\alpha = 0.0167$.

The results for each developer, including the sample sizes for the periods without Copilot (n_1) and with Copilot (n_2), the U statistic, p-value, and Cliff's Delta (δ) for effect size, are as follows:

- **Developer 1:** ($n_1 = 36, n_2 = 27$); U = 972.0; p = 0.001; $\delta = 0.47$ (medium to large effect)
- **Developer 2:** ($n_1 = 14, n_2 = 5$); U = 100.0; p = 0.002; $\delta = 0.52$ (large effect)
- **Developer 3:** ($n_1 = 5, n_2 = 5$); U = 350.0; p = 0.001; $\delta = 0.31$ (small to medium effect)

All p-values are statistically significant under the Bonferroni-corrected level [4] ($p < 0.0167$). The effect sizes indicate a practical significance ranging from small-to-medium (Developer 3) to large (Developer 2). This suggests that while the increase in the code change rate with Copilot was consistently observed, its practical magnitude varied among the participants.

In summary, the individual analysis suggests a positive trend of performance improvement for the participants. All three developers showed a statistically significant increase in their code change rate with the use of GitHub Copilot, a finding that holds even under a conservative Bonferroni correction. The practical magnitude of this effect, as measured by Cliff's Delta, ranged from small-to-medium (Developer 3) to large (Developer 2), suggesting the tool's impact varied in intensity. While these findings reached statistical significance, the small sample sizes for Developer 3 ($n = 5$ in each period) and in the Copilot period for Developer 2 ($n = 5$) warrant caution in generalizing. Nevertheless, the uniform positive direction of these findings suggests a potential positive impact of the tool across the different developer profiles within the studied team.

5.2 Aggregated Code Change Rate of the Team

To understand the impact of GitHub Copilot from a collective perspective, an analysis was conducted considering the code change rate across all tasks. Table 3 presents the average and median values of the code change rate (LOC/h) for the periods with and without the use of Copilot, where 'n' indicates the number of tasks performed per period.

Table 3: Team code change rate (LOC/h) by period

Period	Mean	Median	n
Without Copilot	13.41	8.74	**55**
With Copilot	26.31	21.62	**37**

The results indicate a marked increase in the team's average code change rate after the introduction of Copilot—from 13.41 to 26.31 LOC/h. The median followed a similar upward trend, rising from 8.74 to 21.62 LOC/h, suggesting that the increase was distributed across the tasks and not driven by outliers.

To assess the statistical significance of this difference, the Mann-Whitney U test was applied, yielding the following result:

- **U = 1635.0**; p-value < 0.001

This result confirms that the observed difference between the periods is statistically significant. The effect size, measured using *Cliff's Delta*, was:

- $\delta = 0.53$ (large effect)

These findings suggest a positive impact of GitHub Copilot on the team's overall code change rate. The overall increase in metrics, combined with statistical significance and a large effect size, indicates that the tool contributed to reducing development effort at the team level.

Although effects may vary by developer and task type (as discussed in other subsections), the aggregated analysis reinforces the hypothesis that AI-based coding assistants can offer measurable benefits in real-world work settings.

5.3 Code Change Rate by Task Type

To deepen the understanding of GitHub Copilot's impact on development effort, a segmented analysis was performed by task type. Tasks were grouped into the categories **Bugfix** and **Customization**, as previously described. Table 4 presents the descriptive statistics of the code change rate (LOC/h) for each category during periods with and without the tool, and 'n' indicates the number of tasks performed per period..

Table 4: Summary of code change rate (LOC/h) by task type and period

Type	Period	Mean	Median	n
Bugfix	Without Copilot	38.44	18.87	**30**
Bugfix	With Copilot	48.27	24.18	**20**
Customization	Without Copilot	31.34	16.42	**25**
Customization	With Copilot	13.60	6.86	**17**

The descriptive analysis suggests different trends between the two task types. For **Bugfix** tasks, an increase was observed in both the mean and median code change rates during the Copilot period. This may indicate a positive influence of the tool on tasks related to code correction.

Conversely, tasks related to **Customization** showed a decrease in both average and median rates in the same period. This outcome may be related to the nature of these tasks, which often require a deeper understanding of specific business rules, potentially limiting the effectiveness of automated code suggestions.

To verify the statistical significance of these variations, Mann–Whitney U tests were applied for each task type. The results are presented in Table 5.

Table 5: Statistical test results by task type

Task Type	Mann–Whitney U	p-value	Cliff's Delta
Bugfix	928.0	0.441	0.14 (small)
Customization	100.0	0.152	-0.31 (medium)

For **Bugfix** tasks, the difference between the periods was not statistically significant (p = 0.441), and the effect size was considered small. This suggests that although there was an average gain in the code change rate, Copilot may not have consistently impacted this task type across the entire sample.

For **Customization** tasks, although statistical significance was also not observed (p = 0.152), the *Cliff's Delta* indicated a medium and negative effect size, pointing to a tendency toward a reduction in the code change rate when using the tool. This behavior may reflect the interpretive and contextual complexity of customization tasks, which often require deliberate interventions and are less amenable to direct automation.

In summary, the results of this subsection highlight the importance of considering task type when evaluating the impact of automatic code generation tools. GitHub Copilot may be more effective in structured and standardized tasks—such as many bug fixes—whereas its impact may be more limited, or even counterproductive, in activities that require a high degree of customization or contextual analysis.

5.4 Summary of Quantitative Results

The quantitative analysis of this case study enabled the evaluation of GitHub Copilot's impact from three complementary perspectives: by individual developer, by the team as a whole, and by task type. In all approaches, the metric used was the *code change rate* (LOC/h), interpreted as a proxy for *development effort* and calculated according to Equation 1.

At the individual level, all three developers showed a statistically significant increase in their code change rate during the Copilot usage period, a finding that holds under a Bonferroni correction. The practical magnitude of this effect, measured by Cliff's Delta, varied among the participants, ranging from small-to-medium to large. This suggests that the tool was associated with a reduction in development effort for all individuals, though with differing intensity.

The aggregated team analysis indicated a similar trend. The team's average code change rate increased from 13.41 LOC/h to 26.31 LOC/h (an approximate 96% growth). This difference was statistically significant ($p < 0.001$) and accompanied by a large effect size ($\delta = 0.53$), reinforcing the observation of a reduced development effort at the team level with the tool's use.

However, when analyzing the impact segmented by task type, a more nuanced pattern emerged. For *Bugfix* tasks, a positive trend in the code change rate was observed, though it did not reach statistical significance and the effect size was small. In contrast, *Customization* tasks showed a decrease in the rate, also not statistically significant, but with a medium negative effect size ($\delta = -0.31$) suggesting that, for this specific task category, the tool's use was associated with a reduced rate of code changes.

These findings lead to a multi-faceted interpretation. While GitHub Copilot was associated with a statistically significant reduction in development effort at the individual and team levels, this high-level view masks its varied performance on different types of tasks. The positive effect on the pace of development appears more applicable to corrective maintenance (*Bugfix*) than to tasks requiring significant functional adaptation (*Customization*). These aspects are further explored through the developers' perceptions in the following section.

5.5 Qualitative Analysis of Perceptions on GitHub Copilot Use

To complement the quantitative analysis and understand the subjective aspects related to the use of GitHub Copilot, the participating developers completed a perception form. The goal was to capture perceptions regarding the tool's usefulness, usability, and its impact on daily work, as well as to identify behavioral and cognitive effects associated with its adoption.

The three developers who responded to the questionnaire are experienced professionals, each with over ten years of programming experience, including more than five years of direct involvement with the Integrated Management Systems (SIG) at UFPI. None of the participants had previously used GitHub Copilot in their work

environment, although they were already familiar with the tool. This reinforces the relevance of the study as a first formal adoption experience in a real-world context.

Table 6 presents a summary of the responses to the closed-ended questions included in the questionnaire.

The analysis of the responses reveals a positive perception of the tool. All participants stated that Copilot influenced their way of thinking or problem-solving, suggesting that the assistant functions not only as a code generator but also as a cognitive support tool for organizing and structuring ideas during programming. This perception is consistent with the open-ended responses, which described the tool as a "reasoning assistant."

All developers also expressed interest in continuing to use Copilot, indicating a good level of acceptance of the tool despite the technical limitations of the development environment (Eclipse). However, only one out of the three respondents stated they would be willing to pay for a monthly license, while the other two conditioned its use on institutional availability. This suggests that, although the tool is well received, cost may represent a barrier to continued adoption in public or budget-constrained contexts.

From a technical standpoint, one participant reported difficulties in installing and configuring the Copilot4Eclipse extension, requiring environment updates and additional support. This initial barrier highlights the importance of more integrated and accessible solutions to foster the adoption of AI-based tools in traditional development environments, especially on legacy platforms.

Regarding the open-ended responses, developers reported that Copilot provided useful suggestions, particularly in *bugfix* tasks, where standard structures and syntactic repetitions are common. In contrast, for *customization* activities, the tool performed less effectively, often offering less relevant suggestions and requiring more manual effort to adapt to the system's context.

In summary, the qualitative analysis reveals that while Copilot does not replace the developer's expertise, it is perceived as a useful and complementary resource, with greater impact on predictable and lower-complexity tasks. Overall acceptance is positive but conditioned by factors such as cost, ease of integration, and technological context. These findings reinforce the idea that the successful adoption of AI-assisted code generation tools depends not only on their technical performance but also on users' perceived value and the compatibility with their working environment.

Answer to the Research Question

Based on the quantitative and qualitative evidence presented, it is possible to answer the central question of this study: *How does the use of GitHub Copilot impact the performance and perception of professional developers in a real-world software development environment?* The data indicate that the tool's use is associated with an increase in the rate of code changes in real tasks, especially among more experienced developers and in lower-complexity activities. Moreover, participants' perceptions reinforce that Copilot served as a helpful aid in daily work, offering suggestions that sped up coding and encouraged different solution approaches. These findings suggest that, although Copilot does not replace prior knowledge or technical expertise, it can act as a catalyst in the development process when used in a contextualized and critical manner.

6 Lessons Learned

The practical experience of introducing GitHub Copilot into a professional environment for maintaining legacy systems led to the identification of several lessons that may guide similar initiatives in other organizations. The main lessons learned were:

- **Technical integration is a challenge:** the adoption of AI-based tools depends on compatible infrastructure. The need to update the development environment (Eclipse) and configure specific extensions posed a significant initial barrier.
- **The learning curve is low, but technical judgment is essential:** although developers adapted quickly to using the tool, its effective use required critical evaluation of suggestions and prior system knowledge.
- **Copilot serves as cognitive support:** the tool was perceived as useful for organizing ideas and initiating code snippets, acting as a "reasoning assistant" at the start of implementation.
- **Tool acceptance was positive but depends on institutional factors:** all participants expressed interest in continuing to use Copilot, although its ongoing use depends on the financial and institutional feasibility of licensing.
- **The tool's impact varies by task type:** the data indicated greater effectiveness in repetitive or standardized tasks, such as simple bug fixes, while its performance was more limited in complex customizations.
- **Case studies require methodological flexibility:** conducting research in a real-world environment required adaptations to the schedule, data collection, and analysis, emphasizing the importance of aligning with participants' routines and demands.

These lessons highlight the potential of GitHub Copilot as a support tool for development, while also underscoring the need for careful contextualization for its effective adoption.

7 Threats to Validity

The validity of this study may be affected by several factors that should be considered when interpreting the results. The practical

Table 6: Summary of developers' responses to closed-ended questions

Question	Yes	No	Other
Had you used Copilot before?	0	3	0
Did you experience technical difficulties during installation?	1	2	0
Did Copilot influence your way of thinking?	3	0	0
Would you like to continue using it?	3	0	0
Would you pay for a license?	1	2	0

challenges faced during the study are discussed here as they relate to the threats categorized as internal, external, construct, and conclusion validity.

Internal Validity: Internal validity may have been compromised by several uncontrolled variables. These include the developers' autonomy in deciding when to use GitHub Copilot, individual differences in motivation, and the varying complexity of tasks assigned during the observation periods. Furthermore, initial technical challenges with updating the development environment (Eclipse) may have influenced the initial interactions with the tool. Although all participants belonged to the same team, these factors inherent to an observational study in a real-world setting may have influenced the results.

External Validity: The generalization of the results should be approached with caution. The study was conducted within a specific team at a single public institution, with its own unique culture, processes, and legacy technologies. The choice of organization was influenced by practical considerations, such as license availability through academic programs. These specific contextual factors limit the direct replicability of the findings in other organizational settings, such as private companies, startups, or teams using different development methodologies.

Construct Validity: The primary threat to construct validity lies in the metrics used. Specifically, the primary metric, *code change rate (LOC/h)*, serves as a proxy for development effort, not as a direct measure of productivity or value delivered. This metric can be influenced by factors such as automated code refactoring suggested by Copilot, which may not be strictly necessary for task completion. While our analysis assumes that the code changes are related to the task (as they were validated and committed), this potential for extraneous changes represents a limitation on the interpretation of our quantitative results. Additionally, challenges in integrating and verifying data from different sources (the task management system and Git repositories) may have introduced inaccuracies in the measurement of task execution time and code volume.

Conclusion Validity: The main threat to conclusion validity is the small sample size. Statistical analysis was conducted with only three developers, which affects the statistical power and the robustness of inferences. Although measures of central tendency, dispersion, and non-parametric tests appropriate for the data were applied, the unequal distribution of tasks between the periods and the limited number of participants mean the results should be interpreted as exploratory evidence, highlighting a trend that merits further investigation in larger, more diverse contexts.

8 Conclusion

This paper presented an experience report based on a case study conducted with professional developers from the Superintendence of Information Technology at UFPI, aimed at investigating the effects of using GitHub Copilot in a real-world software development environment. Over a three-month observation period, both quantitative and qualitative data were collected, allowing the tool to be analyzed from multiple perspectives, including code change rate, task type, and developers' perceptions.

Quantitative results indicated an overall increase in the code change rate for tasks performed with Copilot support. However, this effect varied by task type. While a positive trend was observed for bug-fixing tasks, the tool's impact was more nuanced for customization tasks, where the code change rate showed a decrease, suggesting these tasks may require a deeper contextual understanding that limits the benefits of automated suggestions.

From a qualitative standpoint, participants reported improvements in agility and coding support, although they emphasized the importance of critical judgment when interpreting and validating the tool's suggestions. The analysis of their responses reinforced the notion that Copilot serves as an additional support, particularly useful during the initial phases of coding, but does not replace technical knowledge or familiarity with the application context.

Despite promising results, this study has limitations. The sample was limited to only three developers, and the institutional context carries specific characteristics that constrain the generalization of findings. Furthermore, the study did not directly evaluate code quality metrics or collaborative aspects of development. Furthermore, our qualitative analysis did not include specific questions targeting developers' perceptions of trust and accuracy in the tool's suggestions, which are important dimensions of AI-assisted development.

For future work, the following directions are suggested:

- Extend the study to teams with more diverse profiles and across different organizations, to validate the findings in broader contexts.
- Investigate Copilot's impact on full development cycles, including testing, code review, and continuous integration phases.
- Analyze the tool's influence on the quality of the produced code, using automated metrics and peer reviews.
- Explore the role of the tool in collaborative tasks and in pair programming scenarios.
- Explore in depth the developers' perceptions of trust and accuracy regarding AI-generated suggestions, investigating how these factors evolve over time and influence tool adoption.

It is concluded that, although it does not represent a universal solution, GitHub Copilot has shown potential to support developers in executing specific tasks, contributing to a more fluid coding process in professional settings.

REFERENCES

[1] Gal Bakal, Ali Dasdan, Yaniv Katz, Michael Kaufman, and Guy Levin. 2025. Experience with GitHub Copilot for Developer Productivity at Zoominfo. arXiv:2501.13282 [cs.SE] https://arxiv.org/abs/2501.13282

[2] Laurence Bardin. 1977. *Análise de Conteúdo.* Edições 70, Lisboa.

[3] Samuel Silvestre Batista, Bruno Branco, Otávio Castro, and Guilherme Avelino. 2024. Code on Demand: A Comparative Analysis of the Efficiency Understandability and Self-Correction Capability of Copilot ChatGPT and Gemini. In *Proceedings of the XXIII Brazilian Symposium on Software Quality (SBQS '24).* Association for Computing Machinery, New York, NY, USA, 351–361. doi:10.1145/3701625.3701673

[4] J Martin Bland and Douglas G Altman. 1995. Multiple significance tests: the Bonferroni method. *Bmj* 310, 6973 (1995), 170.

[5] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. arXiv:2005.14165 [cs.CL] https://arxiv.org/abs/2005.14165

[6] Sergio Cavalcante, Erick Ribeiro, and Ana Oran. 2025. The Impact of AI Tools on Software Development: A Case Study with GitHub Copilot and Other AI Assistants. In *Proceedings of the 27th International Conference on Enterprise Information Systems - Volume 2: ICEIS.* INSTICC, SciTePress, 245–252. doi:10.5220/0013294700003929

[7] Sayan Chatterjee, Ching Louis Liu, Gareth Rowland, and Tim Hogarth. 2024. The Impact of AI Tool on Engineering at ANZ Bank: An Empirical Study on GitHub Copilot within a Corporate Environment. *arXiv preprint arXiv:2402.05636* (2024). https://arxiv.org/abs/2402.05636

[8] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harrison Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating Large Language Models Trained on Code. (2021). arXiv:2107.03374 https://arxiv.org/abs/2107.03374 Disponível em https://arxiv.org/abs/2107.03374.

[9] Vincenzo Corso, Leonardo Mariani, Daniela Micucci, and Oliviero Riganelli. 2024. Assessing AI-Based Code Assistants in Method Generation Tasks. In *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings* (Lisbon, Portugal) *(ICSE-Companion '24).* Association for Computing Machinery, New York, NY, USA, 380–381. doi:10.1145/3639478.3643122

[10] Nicole Davila, Igor Wiese, Igor Steinmacher, Lucas Lucio da Silva, Andre Kawamoto, Gilson Jose Peres Favaro, and Ingrid Nunes. 2024. An Industry Case Study on Adoption of AI-based Programming Assistants. In *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice* (Lisbon, Portugal) *(ICSE-SEIP '24).* Association for Computing Machinery, New York, NY, USA, 92–102. doi:10.1145/3639477.3643648

[11] GitHub. 2021. Introducing GitHub Copilot: your AI pair programmer. *GitHub Blog* (2021). Available at: https://github.blog/2021-06-29-introducing-github-copilot-ai-pair-programmer/.

[12] GitHub. 2023. GitHub Copilot – November 30th Update. https://github.blog/changelog/2023-11-30-github-copilot-november-30th-update/. Accessed: December 4, 2024.

[13] Rasha Ahmad Husein, Hala Aburajouh, and Cagatay Catal. 2025. Large language models for code completion: A systematic literature review. *Computer Standards & Interfaces* 92 (2025), 103917. doi:10.1016/j.csi.2024.103917

[14] Majeed Kazemitabaar, Justin Chow, Carl Ka To Ma, Barbara J. Ericson, David Weintrop, and Tovi Grossman. 2023. Studying the effect of AI Code Generators on Supporting Novice Learners in Introductory Programming. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems* (Hamburg, Germany) *(CHI '23).* Association for Computing Machinery, New York, NY, USA, Article 455, 23 pages. doi:10.1145/3544548.3580919

[15] John R Levine, Tony Mason, and Doug Brown. 1992. *Lex & yacc.* " O'Reilly Media, Inc.".

[16] Werney Lira, Pedro Santos Neto, and Luiz Osorio. 2024. Uma análise do uso de ferramentas de geração de código por alunos de computação. In *Anais do IV Simpósio Brasileiro de Educação em Computação* (Evento Online). SBC, Porto Alegre, RS, Brasil, 63–71. doi:10.5753/educomp.2024.237427

[17] Microsoft. 2024. IntelliCode. https://learn.microsoft.com/en-us/visualstudio/ide/using-intellisense. Acessado: 13 de jun. de 2025.

[18] Microsoft. 2024. Visual Studio Code - Code Editing. Redefined. https://code.visualstudio.com/. Acessado: 13 de jun. de 2025.

[19] Mauricio Monteiro, Bruno Castelo Branco, Samuel Silvestre, Guilherme Avelino, and Marco Tulio Valente. 2025. NoCodeGPT: A No-Code Interface for Building Web Apps With Language Models. *Software: Practice and Experience* 55, 8 (2025), 1408–1424. arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.3432 doi:10.1002/spe.3432

[20] JunSeong Moon, RaeEun Yang, SoMin Cha, and Seong Baeg Kim. 2023. chatGPT vs Mentor : Programming Language Learning Assistance System for Beginners. In *2023 IEEE 8th International Conference On Software Engineering and Computer Systems (ICSECS).* 2023 IEEE 8th International Conference On Software Engineering and Computer Systems (ICSECS), Penang, Malaysia, 106–110. doi:10.1109/ICSECS58457.2023.10256295

[21] 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. doi:10.1016/j.jss.2023.111734

[22] 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* (Pittsburgh, Pennsylvania) *(MSR '22).* Association for Computing Machinery, New York, NY, USA, 1–5. doi:10.1145/3524842.3528470

[23] Fernando Osorio. 2025. Dataset: Assessing the Use of a Code Generation Assistant in Professional Software Development: An Experience Report. https://github.com/fernandoosorio/sbqs2025. Disponível em repositório GitHub.

[24] Luiz Osorio, Pedro Santos Neto, Guilherme Avelino, and Werney Lira. 2025. An Evaluation of the Impact of Code Generation Tools on Software Development. In *Anais do XXI Simpósio Brasileiro de Sistemas de Informação* (Recife/PE). SBC, Porto Alegre, RS, Brasil, 625–634. doi:10.5753/sbsi.2025.246605

[25] 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). Available at https://arxiv.org/abs/2406.17910.

[26] Sida Peng, Eirini Kalliamvakou, Peter Cihon, and Mert Demirer. 2023. The Impact of AI on Developer Productivity: Evidence from GitHub Copilot. arXiv:2302.06590 [cs.SE] https://arxiv.org/abs/2302.06590

[27] James Prather, Brent N. Reeves, Paul Denny, Brett A. Becker, Juho Leinonen, Andrew Luxton-Reilly, Garrett Powell, James Finnie-Ansley, and Eddie Antonio Santos. 2023. "It's Weird That it Knows What I Want": Usability and Interactions with Copilot for Novice Programmers. *ACM Trans. Comput.-Hum. Interact.* 31, 1, Article 4 (Nov. 2023), 31 pages. doi:10.1145/3617367

[28] Yashaswini Raghavan. 2023. The Impact of GitHub Copilot on Developer Productivity: A Case Study. *Harness Software Engineering Insights* (2023). https://harness.io/blog/the-impact-of-github-copilot-on-developer-productivity-a-case-study

[29] Forrest Shull, Janice Singer, and Dag IK Sjøberg. 2008. *Guide to Advanced Empirical Software Engineering.* Vol. 93. Springer, Berlin, Heidelberg.

[30] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2023. Attention Is All You Need. arXiv:1706.03762 [cs.CL]

[31] Roel Wieringa and Ayse Morali. 2012. Technical action research as a validation method in information systems design science. In *International Conference on Design Science Research in Information Systems.* Springer, Berlin, Heidelberg, 220–238.

[32] Claes Wohlin and Aybüke Aurum. 2015. Towards a decision-making structure for selecting a research design in empirical software engineering. *Empirical Softw. Engg.* 20, 6 (Dec. 2015), 1427–1455. doi:10.1007/s10664-014-9319-7

[33] Han Xue and Yanmin Niu. 2023. Exercise Generation and Student Cognitive Ability Research Based on ChatGPT and Rasch Model. *IEEE Access* 11 (2023), 1–1. doi:10.1109/ACCESS.2023.3325741

[34] Burak Yetiştiren, Işık Özsoy, Miray Ayerdem, and Eray Tüzün. 2023. Evaluating the Code Quality of AI-Assisted Code Generation Tools: An Empirical Study on GitHub Copilot, Amazon CodeWhisperer, and ChatGPT. arXiv:2304.10778 [cs.SE] https://arxiv.org/abs/2304.10778

[35] Li Zhang, Jia-Hao Tian, Jing Jiang, Yi-Jun Liu, Meng-Yuan Pu, and Tao Yue. 2018. Empirical Research in Software Engineering — A Literature Survey. *Journal of Computer Science and Technology* 33, 5 (September 2018), 876–899. doi:10.1007/s11390-018-1864-x