Empirical Evaluation of Developer Productivity Enhancement with Microsoft Copilot Using the SPACE Framework

Team 10: Manoj, Yashwanth, Sravanthika

Introduction

The advent of AI-driven code generation tools, such as Microsoft Copilot, has brought a paradigm shift in the software development landscape. While these tools hold significant potential, it is crucial to empirically understand the real-world experiences of developers to optimize their usability and positively impact developer productivity.

Purpose of the Study

This study is designed to evaluate the impact of Microsoft Copilot on developer productivity across various dimensions. Specifically, the research aims to:

- Assess how Microsoft Copilot affects developer satisfaction with its ease of use and perceived usefulness.
- Determine whether Copilot enhances coding efficiency and effectiveness, thereby improving overall developer performance.
- Investigate the integration of Microsoft Copilot into diverse coding activities, identifying both strengths and areas for improvement.

Background Research

Microsoft Copilot, powered by OpenAI's GPT technology, is a state-of-the-art code generation tool that assists developers in writing code snippets, suggesting code completions, and providing contextual code examples. It aims to streamline coding tasks and improve developer efficiency. However, user feedback and previous research have indicated several challenges and pain points. [1]

Previous studies and user reports have pointed out issues such as non-intuitive suggestions and difficulties integrating with complex existing codebases. These insights underscore the need for a structured evaluation using the SPACE Framework, which focuses on Satisfaction and Well-being, Performance, Activity, Communication and Collaboration, and Efficiency and Flow. This framework will guide our systematic analysis to draw comprehensive conclusions about Copilot's real-world utility and to develop actionable recommendations for enhancing its deployment in software development environments. [2]

Research Questions

1. How does Microsoft Copilot influence developer satisfaction regarding ease of use and perceived usefulness?
2. Does using Microsoft Copilot enhance developer performance regarding coding efficiency and effectiveness?
3. How seamlessly does Microsoft Copilot integrate into the coding activities of developers?

Concepts

Research	Exploratory
Concepts	Satisfaction, Performance, and Activity.
Dimensions	Ease of use, Perceived usefulness, Effectiveness, Efficiency.
Type of Experiment	Non-experiment
Unit of Analysis	Individuals (Software developers)
Type of study (Time)	Longitudinal

Operationalization

Developer Satisfaction: Measured through self-reported survey responses. This variable captures developers' assessments of Microsoft Copilot in terms of its ease of use and overall satisfaction.

Developer Performance: This variable is Operationalized by analyzing several key metrics including the intensity of its usage (time spent interacting with the Copilot), and the number of revocations of Copilot's suggestions as well as syntax errors. These metrics collectively provide a comprehensive picture of the tool's impact on coding efficiency and effectiveness.

Developer Activity: This aspect is operationalized through the measurement of the total number of lines of code written during sessions that involve Copilot tasks.

Measures and Instruments

Concepts	Dimension	Measure	Instrument
Developer Satisfaction	Ease of use and Perceived usefulness	Likelihood of frequent usage, Perception of complexity	SPACE Survey Audio recordings
Developer Performance	Effectiveness and Efficiency	Usage intensity, Number of syntax errors, Number of revokes	Logs – Custom log script Audio recordings
Developer Activity(Unidimensional Concept)		Number of lines of code	Code Snapshots Audio recordings

Methods

Mixed Methods Research Design

Overview of Design Choice:

This study employs a Convergent Parallel Mixed Methods Design to investigate the impact of Microsoft Copilot on software development. This approach was selected for its robustness in combining and corroborating both quantitative and qualitative data to provide a comprehensive analysis of the research questions.

Implementation in the Study:

Quantitative data were collected through SPACE surveys, logs, and code snapshots to measure variables such as usability, coding efficiency, and error rates. Simultaneously, qualitative data were gathered from semi-structured interviews, offering insights into developers' perceptions and experiences with Copilot. These two strands of data were collected concurrently and analyzed independently.

Integration of Data:

After separate analyses, findings from both the quantitative and qualitative strands were integrated during the interpretation phase. This integration allowed for a deeper understanding of how numerical trends from the survey and logs correlate with the personal experiences and feedback provided by the developers during interviews. This method not only enriches the findings but also provides a multi-dimensional perspective on the effectiveness of Copilot in real-world coding environments.

Justification for Mixed Methods:

The use of a mixed methods design enhances the study's validity by allowing for cross-validation where quantitative data's breadth is complemented by qualitative insights' depth. This approach is particularly beneficial in exploring new or complex phenomena where different data types can illuminate different aspects of the research questions.

Validity of Research Instruments

Content Validity:

The SPACE survey and developer logs are designed based on established theories and empirical studies, ensuring they accurately capture relevant aspects of software development like satisfaction and performance. This design guarantees that these instruments are aligned with the study's objectives, such as evaluating Copilot’s impact on developer productivity.

Construct Validity:

Construct validity is reinforced through the triangulation of quantitative (surveys and logs) and qualitative (interviews) data, ensuring consistent measurement of constructs like usability and productivity across different methods. Hypothesis testing, including Pearson correlation and Repeated Measures ANOVA, validates the theoretical relationships and effects, enhancing construct validity.

Reliability

Reliability for the SPACE survey is verified by calculating Cronbach's alpha, with a high value indicating robust internal consistency. Similarly, the reliability of logs and code snapshots is ensured through regular validation checks that confirm accurate and consistent data capture.

Participants

This study involved a total of four participants, comprised of three males and one female. All participants are experienced software developers employed at various IT companies and have prior familiarity with AI-based tools. The participants were selected through a demographic survey that also gathered information on their satisfaction with current AI coding tools and the average daily time spent coding. Inclusion in the study was based on their consent to participate, ensuring they were informed about the research's scope and the use of their data. This method ensured that all participants were both knowledgeable about the topic and willing to contribute to the research findings. [Demographics Survey Link]

Data Collection

SPACE Survey

The data for this study was collected using the SPACE survey, which is designed to comprehensively assess various aspects of developer interaction with Microsoft Copilot. The survey is divided into two sections:

SUS Scale (Questions 1-10): This section focuses on Developer Satisfaction, specifically evaluating the ease of use and perceived usefulness of Microsoft Copilot.

LTR Scale (Question 11 -12): This part of the survey measures the likelihood that participants would recommend Microsoft Copilot to others, based on their usage experience.

Data was collected at two-time points, enabling a longitudinal study of developers' perceptions and interactions with Microsoft Copilot. This method allows for analyzing how developers' views and usage of Copilot evolve, providing insights into its long-term impact on their productivity and workflow. [SPACE Survey Link]

Logs and Code Snapshots

We developed a custom logging script to record the intensity of usage, specifically measuring the time developers interacted with Microsoft Copilot. The script also tracked the number of times developers revoked Copilot's suggestions and recorded syntax errors, providing insights into the tool's effectiveness and efficiency. Additionally, developers submitted code snapshots during their sessions with Copilot, allowing us to analyze the volume of code produced, which reflects their activity levels.

Code Snapshots – [Code Snapshots from Developers.pdf]

Semi Structured Interviews

For the Semi-Structured Interviews, open-ended questions were crafted based on the insights gleaned from the Quantitative results. Through open-ended questions, we delved into their initial impressions, challenges, and evolving perceptions of the tool. These qualitative insights enriched our understanding of how Copilot integrates into developers' workflows and its potential impact on coding practices, contributing depth and context to our research findings.

Interview Script – [Interview Script.pdf]

Data Analysis

Quantitative Analysis

Developer Satisfaction

We analyzed the System Usability Scale (SUS) Likert responses, and we first transformed these responses into numerical values. For items with odd numbers, we subtracted one from each user response, and for even-numbered items, we subtracted the user responses from five. This method adjusted all values to range from 0 to 4, where 4 indicates the most positive response. We then summed the adjusted responses for each participant and multiplied the total by 2.5, thus converting the range of possible values from 0 to 100, rather than 0 to 40. The processed data were entered into an Excel sheet and analyzed in R Studio for descriptive statistics, allowing us to derive SUS scores, learnability, and likelihood to recommend (LTR) scores comprehensively.

https://rpubs.com/paulamat/sus - SUS scores

https://measuringu.com/nps-sus/ - LTR scores

SPACE Survey 1.xlsx - Survey Responses

Developer Performance and Developer Activity

A custom logging script was utilized to record key metrics such as usage intensity, the number of syntax errors, and the number of times users revoked Copilot's suggestions. This data was formatted into JSON and processed through a Logstash pipeline to filter and parse the data effectively, extracting crucial elements like timestamps, duration of use, and the number of revocations to assess effectiveness and efficiency.

Custom Log.csv - Usage_Intensity_Syntax_Errors.xlsx

Code Snapshots were also taken to measure the number of lines of code produced during interactions with Copilot. To further enhance our analysis, a Python script was developed to extract the number of lines of code from these images, enabling a comprehensive evaluation of developer activity.

Justification of Quantitative Analysis Techniques

Hypothesis Testing:

The quantitative phase of our study was driven by two key hypotheses:

First Hypothesis (Correlation between SUS scores and recommendation likelihood):

Null Hypothesis (H0): There is no significant correlation between perceived usability and the likelihood to recommend Microsoft Copilot among users.

Alternative Hypothesis (H1): There is a significant positive correlation between perceived usability and the likelihood to recommend Microsoft Copilot among users.

Justification for Pearson Correlation:

Choice of Test: Pearson correlation was selected for its precision in measuring the relationship between two **continuous variables**: SUS scores and the likelihood to recommend Copilot.

Applicability: It is particularly appropriate for testing our first hypothesis as it quantifies the degree of linear relationship.

Pre-Analysis Normality Check: The Shapiro-Wilk test verified that the data conformed to a normal distribution, affirming the suitability of Pearson correlation for our dataset.

Second Hypothesis (Impact of usage intensity on syntax errors):

Null Hypothesis (H0): Usage intensity does not significantly affect the number of syntax errors.

Alternative Hypothesis (H1): Higher usage intensity significantly reduces the number of syntax errors.

Justification for Repeated Measures ANOVA:

Independent variables and Dependent Variables

IV: Usage intensity of Microsoft Copilot

Levels: Three levels- Categorical – High, Low, Medium

DV: Number of Syntax Errors

Design of the Experiment – Within Subject Design

Choice of Test: Repeated Measures ANOVA was the statistical test of choice to examine the second hypothesis. This test is adept at analyzing data where the same subjects are measured multiple times under different conditions—in our case, varying levels of Copilot usage intensity.

Applicability: It enabled us to observe the within-subjects effects and interactions over time, which is crucial for assessing the learning curve and long-term performance improvements with Copilot.

Assumption Checks: Sphericity, an assumption of Repeated Measures ANOVA, was tested using Mauchly's test. The assumption was not violated, confirming that Repeated Measures ANOVA was an appropriate choice for our analysis**.**

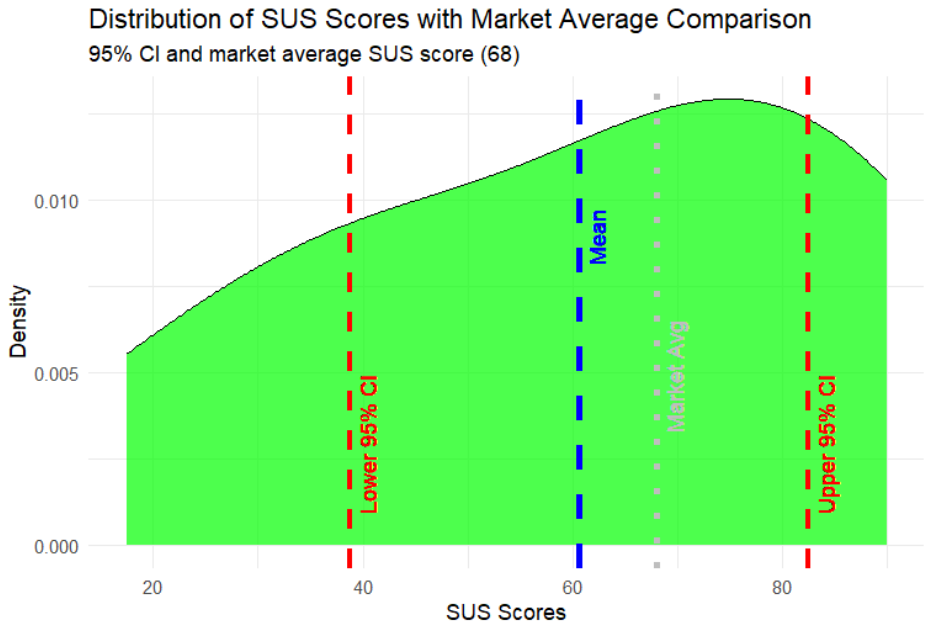
Post hoc tests: Post-hoc analysis was conducted after the main study to identify specific group differences, refining initial findings for a more nuanced understanding. It determines significant differences in error rates across usage intensities, providing crucial insights for optimizing code quality and developer performance.

Results and Findings of Quantitative Analysis

First Hypothesis - First_Hypo.html

Descriptive Stats

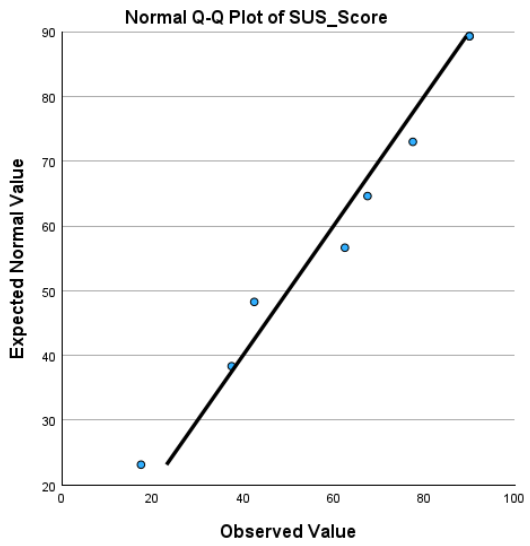
Descriptive statistics reveal that the average SUS score is 60.625 with a standard deviation of 26.1435, indicating varied user perceptions of usability among the eight participants. The LTR scores average 30.3059 with a higher variance, suggesting a broad disparity in willingness to recommend. These findings underscore the importance of usability in influencing recommendation likelihood, aligning with our hypothesis**.**



Normality Test

SUS scores are normally distributed (W = 0.93502, p-value = 0.5628), suitable for parametric tests like Pearson's correlation.

The Pearson correlation analysis was conducted to examine the relationship between perceived usability (measured by SUS scores) and likelihood to recommend (measured by LTR scores) in the dataset.



Correlation Findings

The Pearson correlation coefficient is 0.68, indicating a moderately strong positive relationship between the usability of a product (SUS) and users' likelihood to recommend it (LTR). This suggests that usability improvements are likely to enhance user recommendations.

Statistical Significance

The statistical robustness of this correlation is underscored by a p-value of 4.4975e-08, which allows us to reject the null hypothesis confidently. The analysis provides strong evidence that the correlation is not due to chance.

Confidence Interval

The 95% confidence interval for the correlation coefficient spans from 0.5 to 0.81. This range, not including zero, further validates the significance and strength of the correlation.

Visualization Insights

The scatter plot illustrates a consistent upward trend in recommendation likelihood with increased usability, with a few notable outliers. These exceptions suggest areas for potential further investigation to understand anomalies in user behavior.

Conclusion

The findings emphasize the critical role of usability in influencing recommendations. Enhancing usability should be a key focus for improving user advocacy, supported by continued research into specific usability factors and outlier analysis.

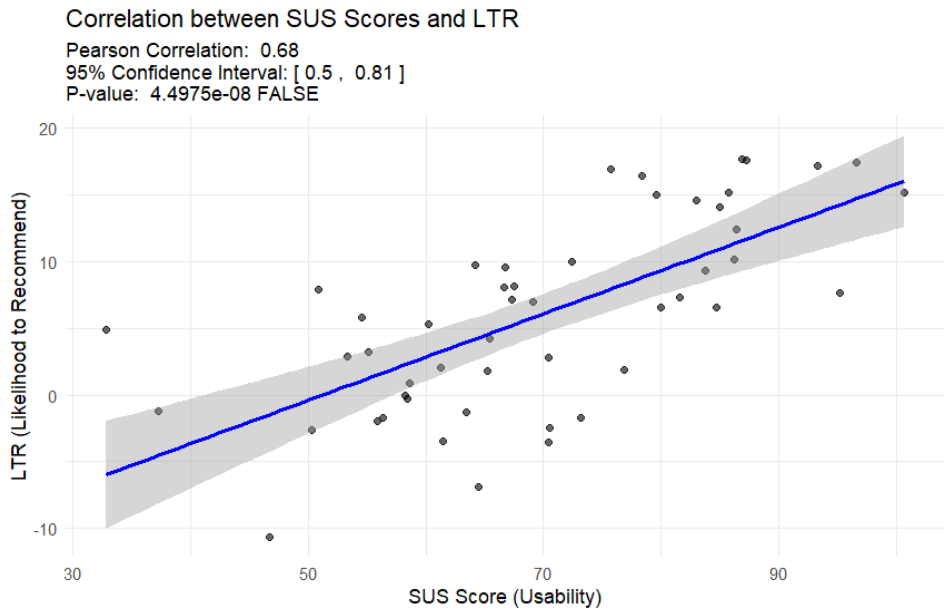


Fig: Relationship between SUS and LTR Scores

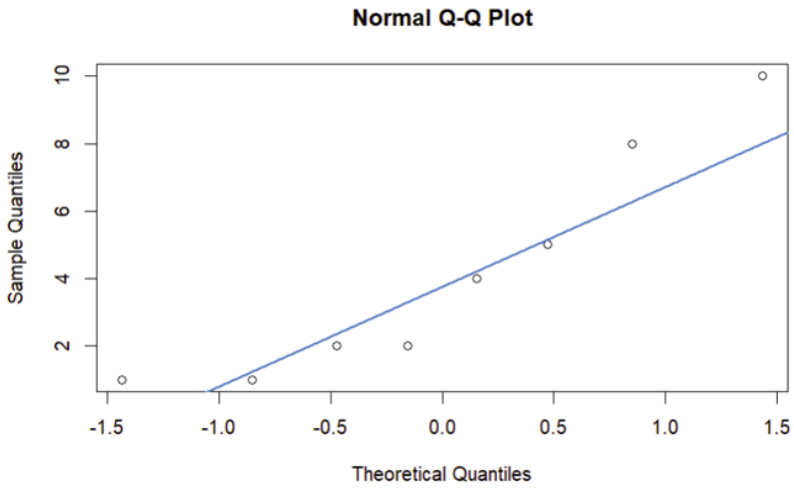
Second Hypothesis

Descriptive Statistics. - SPACE1.nb.html

Descriptive statistics offer quantitative insights into syntax errors, indicating higher mean errors at high usage intensity (8.35) versus lower (2.94) and medium (4.67) intensities. Variability, reflected by similar standard deviations (1.24 to 1.46) across groups, suggests consistent error dispersion regardless of intensity. This numerical breakdown helps identify operational strengths and weaknesses, informing optimizations.

Normality Test

The Q-Q plot with a normal curve, alongside the Shapiro-Wilk test (W = 0.87298, p-value = 0.1612), suggests that syntax errors are normally distributed, with minor deviations at the tails. Although the data show some skewness, they do not significantly deviate from normality, supporting the use of parametric tests.



Repeated Measures ANOVA - Second hypothesis.nb.html

The ANOVA results highlight significant effects of Usage Intensity on syntax errors. The intercept and usage intensity variations show significant F-values (124.45455 and 12.60656 respectively), indicating strong differences in syntax errors across levels of usage intensity.

Mauchly's Test for Sphericity shows no significant violation (W = 0.1268476, p = 0.1268476), affirming the appropriateness of the ANOVA model used.

Sphericity corrections using Greenhouse-Geisser and Huynh-Feldt adjustments confirm the significance of the results, adjusting p-values to 0.03385879 and 0.02814234, respectively, suggesting robust effects across different usage intensities.

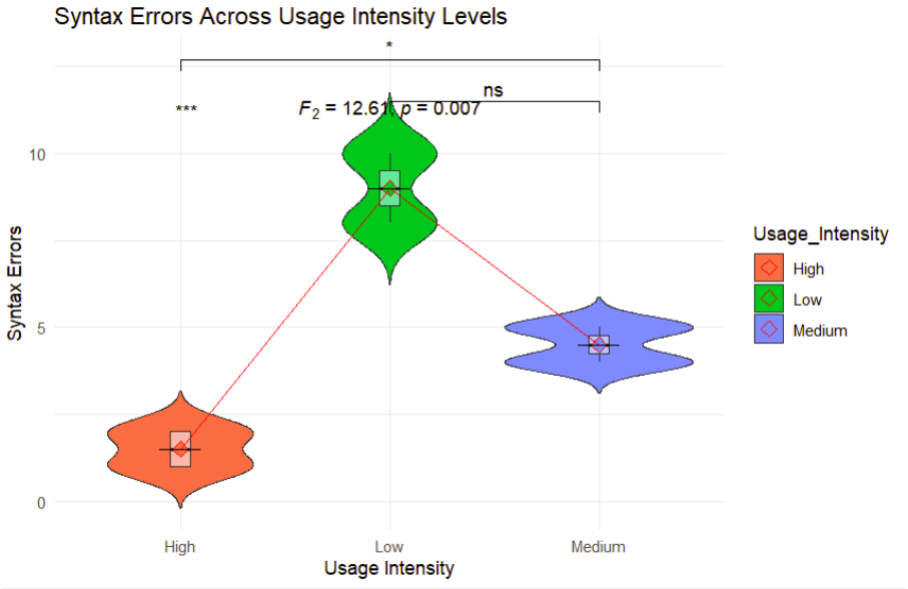


Fig: Effect of Usage Intensity on Syntax Errors

The significant results of the ANOVA affirm that higher engagement with Microsoft Copilot is beneficial in reducing the number of syntax errors made by developers. This finding validates the alternative hypothesis and highlights the practical benefits of increasing the usage intensity of Copilot in coding environments.

Post hoc Tests

Based on the Bonferroni-corrected pairwise comparisons:

There is a significant difference in syntax errors between the "High" intensity and "Low" intensity groups (p = 0.0004345).

There is no significant difference in syntax errors between the "High" intensity and "Medium" intensity groups (p = 0.0269808).

There is a significant difference in syntax errors between the "Low" intensity and "Medium" intensity groups (p = 0.0089817).

Conclusion

The analysis of pairwise comparisons, adjusted using the Bonferroni correction, reveals significant differences in syntax errors between the 'High' and 'Low' intensity groups and between the 'Low' and 'Medium' intensity groups. However, there is no significant difference in syntax errors between the 'High' and 'Medium' intensity groups.

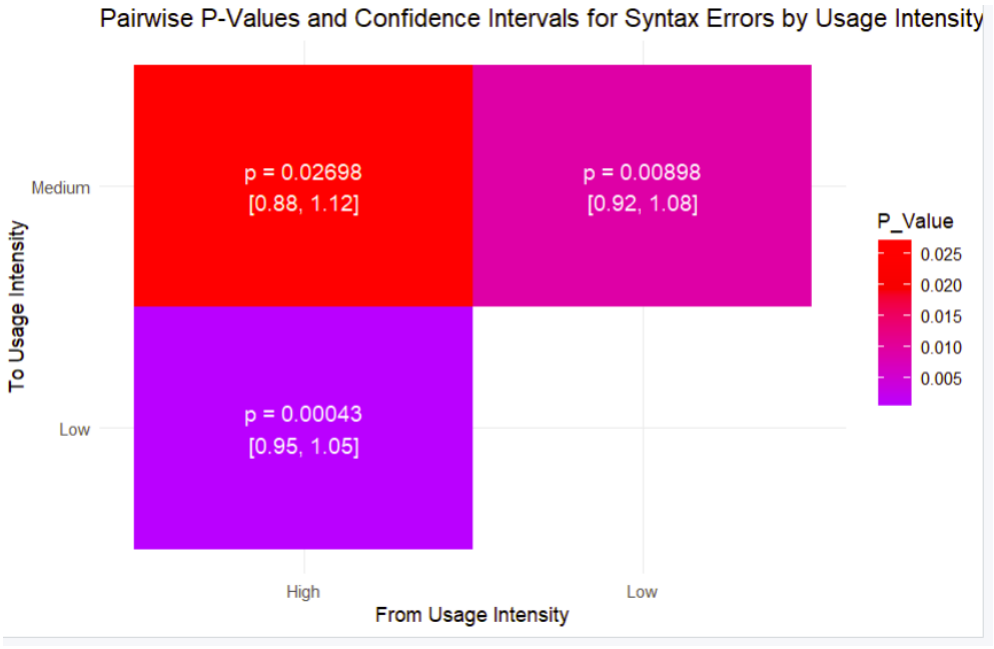


Fig: Post hoc Heatmap

Qualitative Analysis

Objective

The qualitative analysis aimed to deeply understand developers' experiences and perceptions while using Microsoft Copilot, complementing the quantitative findings from the SPACE survey.

Interview scripts were collected through Audio Recordings from four developers who use Microsoft Copilot in their coding tasks. These scripts provided detailed insights into the developers' perspectives.

Interview Responses - Interviews

Thematic Analysis:
Thematic analysis via Atlas.ti facilitated the extraction of core themes from developer interviews regarding their experiences with Microsoft Copilot. It provided a systematic approach to identifying recurring ideas and translating qualitative data into actionable insights.

Justification for Thematic Analysis:
The thematic analysis approach is substantiated by its capacity to distill complex narratives into identifiable patterns. It allows for the qualitative data's richness to be harnessed, offering a detailed understanding of user experiences that extend beyond quantitative measures.

Justification for Codebook Development:
Creating a codebook offered a structured way to categorize and interpret the data. It served to define themes consistently, ensuring that the analysis was both replicable and valid. This approach captured the nuances of developer experiences effectively, including both the strengths and areas for improvement in Copilot's use.
[CS 535 Microsoft Copilot developer experience Code Book (1).xlsx]

Initial Code	Theme
+Ease of use Impact	**PT1: Seamless Integration and Productivity Enhancement**
+Productivity link	
+Coding efficiency	
+Performance metrics	
+Adaptive feature benefit	**PT2: Learning and Adaptation**
+Reduced syntax errors	
+Error rate reduction	
+Positive Experience	**PT3: Usability and recommendation**
+Recommendation to peers	
+Correlation of usability with satisfaction	
+Database integration need	

+Ability to adapt to different coding styles	**PT4: Capability Enhancement Needs**
+Customization process complexity	
+Enhance its capabilities with artificial intelligence	
+Coding effectiveness	**PT5: Effective Use and Impact Measurement**
+Frequency correlation with coding	
+Ease of integration	**PT6: Integration with Coding Activities**
+Excels in handling repetitive data-handling tasks	
-Auto-complete intrusiveness	**NT1: Disruptions and Inefficiencies**
-Coding inefficiency	
-Struggle with more complicated coding	**NT2: Complex coding scenarios**
-Biggest challenge has been keeping up with its updates and new features	**NT3: Challenges with integration**
-Finding the right balance between automation and personal oversight	
-Learning to trust its capabilities	

Fig: Themes Building

Themes Identified

Positive Themes

Seamless Integration and Productivity Enhancement (PT1)

Developers reported that Copilot integrates smoothly into their existing environments, enhancing productivity by automating routine tasks. "The seamless integration allows me to focus more on complex problems rather than syntax, which noticeably boosts my daily output," shared one developer.

Learning and Adaptation (PT2)

Many users appreciated Copilot's ability to adapt to their coding style, enhancing their learning process. "It's like having a smart assistant that learns how you work and gradually starts to anticipate your needs," commented another participant.

Usability and Recommendation (PT3)

The ease of use led to high satisfaction levels among developers, influencing their willingness to recommend Copilot to peers. "Its intuitive nature makes it an easy sell to my colleagues who are always looking for ways to reduce coding errors," a participant noted.

Capability Enhancement Needs (PT4)

Feedback indicated a desire for Copilot to extend its capabilities, particularly in adapting to diverse coding styles and enhancing its artificial intelligence features. "If it could better understand the nuances of my projects, I'd use it even more," one developer suggested.

Effective Use and Impact Measurement (PT3)

Developers linked frequent use of Copilot to significant improvements in their coding efficacy, with many using performance metrics to quantify its impact. "There's a direct correlation between how often I use Copilot and a decrease in my project turnaround times," observed a developer.

Integration with Coding Activities (PT2)

Copilot was praised for its effectiveness in handling repetitive tasks, which was seen as a key factor in its successful integration into daily practices. "For routine data entry and formatting tasks, it's become indispensable," noted a respondent.

Negative Themes

Disruptions and Inefficiencies (NT1)

Some developers reported that Copilot's autocomplete feature could be overly intrusive, occasionally disrupting their coding flow. "There are times when it suggests too much too quickly, which can actually slow me down," a developer expressed.

Complex Coding Scenarios (NT2)

The tool's limitations became apparent in complex coding scenarios, where nuanced understanding is crucial. "In projects requiring intricate logic, Copilot sometimes misses the mark," another participant highlighted.

Challenges with Integration (NT3)

Keeping up with Copilot's frequent updates and features posed challenges for some, impacting their workflow integration. "It's a bit of a double-edged sword; the constant updates are great but keeping up can be demanding," shared a developer.

Conclusion of Qualitative Analysis

The qualitative analysis conducted on the use of Microsoft Copilot among software developers has highlighted significant benefits in terms of coding productivity and the enhancement of learning processes. Developers commend Copilot for its seamless integration into their workflows and its intuitive usability, which collectively contribute to a more efficient coding environment. However, the analysis also identifies ongoing challenges, particularly in handling complex coding scenarios and in coping with frequent software updates. These challenges underscore critical areas where further enhancements are needed.

Additionally, there is a clear demand from the developer community for greater customization options and the incorporation of more advanced artificial intelligence capabilities. Such improvements are necessary to better adapt Copilot to a variety of coding styles and project demands, which would, in turn, enhance its overall effectiveness and user satisfaction.

This study emphasizes the necessity of continuous development and the value of incorporating user feedback to refine and evolve Microsoft Copilot. By addressing these identified needs, Copilot

can be further developed into an even more indispensable tool in the realm of software development, offering profound benefits to its users.

Discussion of Results: Addressing Research Questions with Integrated Insights

Enhanced Precision in Evaluating Microsoft Copilot

This mixed-methods study offers a nuanced assessment of Microsoft Copilot, aiming to resolve specific research questions through a detailed integration of quantitative data from SPACE surveys and developer logs with qualitative insights from semi-structured interviews.

Research Question 1: How does Microsoft Copilot influence developer satisfaction regarding ease of use and perceived usefulness?

Quantitative Insights:

The moderate average SUS score of 60.625, combined with a strong correlation between SUS scores and the likelihood to recommend Copilot (LTR), indicates that while the tool is generally found useful, there is significant room for improvement in its user interface and interaction design to enhance satisfaction.

Qualitative Insights:

Developers consistently praised the tool's seamless integration and the automation of routine tasks, which significantly increased their productivity and satisfaction. However, they also expressed frustrations with Copilot's handling of more complex coding tasks, suggesting a mismatch between Copilot's capabilities and the varied demands of different coding projects.

Integration of Findings:

The integration of these findings pinpoints a critical insight: enhancing Copilot's capabilities in complex scenarios could directly improve user satisfaction and increase its recommendation rates. This suggests a specific pathway for improving the tool's design and functionality.

Research Question 2: Does using Microsoft Copilot enhance developer performance regarding coding efficiency and effectiveness?

Quantitative Performance Metrics:

Data showing a decrease in syntax errors with increased usage intensity of Copilot suggests that the tool significantly contributes to coding efficiency. This relationship highlights Copilot's potential to reduce errors and streamline coding processes.

Qualitative Performance Feedback:

Developers noted that Copilot has become integral in speeding up their routine coding, allowing more time for complex problem-solving. They also reported a noticeable decrease in the frequency and severity of coding errors, affirming the quantitative findings.

Integration of Findings:

These insights reveal that Copilot not only supports developers in managing routine tasks but also enhances their overall coding efficiency. The qualitative feedback underscores the quantitative results, providing a robust basis for claiming that Copilot significantly boosts coding performance.

Research Question 3: How seamlessly does Microsoft Copilot integrate into the coding activities of developers?

Quantitative Integration Data:

The frequent use of Copilot, as shown by usage logs, contrasts with the mixed feedback on its integration efficacy, particularly in complex coding scenarios.

Qualitative Integration Themes:

While the tool is highly valued for routine tasks, developers report challenges in integrating it into more complex coding projects. This feedback points to specific areas where Copilot's functionality could be enhanced to better support a wider range of programming tasks.

Integration of Findings:

This focused analysis demonstrates that while Copilot is effective in certain contexts, its integration into more diverse and complex coding environments requires further development. This precise identification of integration challenges provides a clear direction for future enhancements.

Validity and Generalizability of Findings

Internal Validity:

The rigorous methodological approach, combining detailed quantitative analysis with rich qualitative insights, strengthens the internal validity of the study. The use of established statistical methods and thematic analysis techniques ensures that the findings are well-supported and reliable.

External Validity:

The generalizability of findings is somewhat limited by the small sample size and the specific demographic of developers involved in the study. Future research should aim to include a broader range of participants from various backgrounds and with different levels of experience with AI tools to enhance the external validity and applicability of the findings to a wider developer population.

Justification and Implications for Design

Insights from the Study

Our study has substantiated that Microsoft Copilot can significantly influence developer satisfaction and performance, corroborating the potential of AI-driven tools in enhancing software development practices. The ease of integration into daily coding tasks and its effectiveness in reducing coding errors are particularly noteworthy. These findings not only reinforce the utility of Copilot but also suggest avenues for further enhancement.

Implications for Design

Based on our discussion, analyses, and interviews, several design implications can be formulated to enhance the usability and effectiveness of Microsoft Copilot:

Enhanced Support for Complex Coding Tasks: Given that developers seek better support in complex scenarios, future iterations of Copilot should focus on improving its algorithms to handle intricate coding tasks more effectively.

Customization Options: Developers expressed a desire for more personalized experiences. Introducing more customization options that allow developers to tailor Copilot's behavior to better fit individual coding styles and preferences could enhance its utility.

User Training and Onboarding: To maximize the benefits of Copilot, especially for those less familiar with AI tools, implementing comprehensive training and onboarding sessions could be beneficial. This would help users fully leverage Copilot's capabilities from the outset.

Conclusion

Our research underscores the significant role that Microsoft Copilot can play in enhancing developer productivity and satisfaction. While the tool demonstrates substantial benefits, there is room for growth, particularly in handling complex coding challenges and providing more personalized experiences. As AI technology continues to evolve, further research is needed to explore how these tools can be optimized to better support developers in an increasingly diverse and complex software development landscape.

Serial Number	Team Member	Contributions
1)	Manoj Dattatreya UIN : 659696543 E-mail:mmyne@uic.edu	• Project Proposal • Worked on Research Instruments • Interviewed 2 Participants. • Collected Logs from 2 Participants • Quantitative Findings – Analysis of Inferential Stats and presented Results. • Integration and Discussion of the Results.
2)	Yashwanth Reddy Dasari UIN: 665044614 E-mail:ydasa@uic.edu	• Quantitative Analysis – Descriptive stats and Results of Quantitative Findings • Interviewed 2 Participants. • Collected Logs from 2 Participants • Created a Custom Log Script to Capture Measures related to the Project.
3)	Sravanthika Bandla UIN: 653379708 E-mail:lbandl2@uic.edu	• Performed Qualitative Thematic Analysis and Presented results. • Performed data cleaning and structured the collected data. • Worked on the Project Report and the Presentation. • Collected Data through Surveys From 4 participants.

References

[1] The Impact of AI on Developer Productivity: Evidence from GitHub Copilot by Sida Peng, Eirini Kalliamvakou, Peter Cihon, Mert Demirer.

[2] The SPACE of Developer Productivity: There's more to it than you think. By Nicole Forsgren, Margaret-Anne Storey,Chandra Maddila, Thomas Zimmermann, Brian Houck, And Jenna Butler.

Working Google Document [Data Collection, Findings, Analysis Files]

SPACE UX 535