Human-AI Pair Programming: Evaluating Trust, Efficiency, and Defect Incidence

Arjun Deshraje Urs

arjunursonline@gmail.com

Abstract:
The advent of human-AI pair programming, characterized by collaborative interactions between developers and intelligent code assistants such as GitHub Copilot and Amazon CodeWhisperer, represents a pivotal shift in software engineering practices. This study presents an empirical investigation into the influence of these AI copilots on trust calibration, task efficiency, error proliferation, and the onboarding of software engineers. Utilizing a mixed-methods approach with 72 participants across three experience levels, we conducted a within-subjects experiment comparing solo programming with AI-assisted development. While AI copilots enhance code generation velocity by 19.7% on average, they concurrently increase defect rates by 71% among novice developers ($p < 0.001$). These insights illuminate pathways toward the design of more reliable, interpretable AI assistants and highlight the need for experience-dependent training approaches.

Keywords: AI copilots, collaborative programming, trust calibration, developer efficiency, defect rates, software onboarding

1. INTRODUCTION

The proliferation of AI-powered code generation tools is reshaping software engineering practices. Systems such as GitHub Copilot and Amazon CodeWhisperer function as real-time cognitive partners, dynamically suggesting syntactic and functional code components. Despite their growing adoption, rigorous empirical inquiry into their impact on human cognition, programming efficacy, and defect introduction remains limited. This paper addresses critical gaps in the literature by examining three dimensions of human-AI collaborative programming: trust development, efficiency differentials, and software defect incidence. Drawing on prior research in human-computer interaction and cognitive systems engineering, we hypothesize that developer-AI trust relationships mediate the effectiveness and safety of AI-generated code.

Research Questions

1. How does AI assistance affect programming efficiency across different experience levels?
2. What is the relationship between AI copilot usage and defect introduction rates?
3. How do trust patterns vary among developers of different experience levels when using AI assistance?

2. RELATED WORK

Recent studies have begun exploring the impact of AI coding assistants on developer productivity and code quality. Ziegler and Müller (2022) examined cognitive load implications, while Lee et al. (2023) investigated code quality metrics. However, most existing research lacks comprehensive analysis across experience levels and detailed trust calibration studies. Our work extends this foundation by providing quantitative evidence of experience-dependent effects and introducing novel trust measurement frameworks.

3. METHODOLOGY

3.1 Participants

We recruited 72 professional software developers through industry partnerships and developer communities. Participants were stratified into three experience groups:

- **Group A (Novice)**: 0–2 years experience (n=24, M=1.3 years, SD=0.7)
- **Group B (Intermediate)**: 3–5 years experience (n=24, M=4.1 years, SD=0.8)
- **Group C (Senior)**: 6+ years experience (n=24, M=9.2 years, SD=3.1)

Demographics: 58% male, 40% female, 2% non-binary; representing 15 different programming languages as primary expertise.

3.2 Experimental Design

We employed a within-subjects design where each participant completed two programming tasks under different conditions:

1. **Solo Condition**: Autonomous development without AI assistance
2. **AI-Assisted Condition**: Programming with GitHub Copilot enabled

Task Standardization: Both tasks involved implementing a web API endpoint with authentication, data validation, and database operations. Tasks were equivalent in complexity (verified through pilot testing with 12 additional developers) and counterbalanced to control for order effects.

Bias Control Measures:

- Randomized task order assignment
- 48-hour washout period between conditions
- Different but functionally equivalent requirements for each task
- Participants blind to specific research hypotheses

3.3 Measurement Instruments

3.3.1 Objective Metrics

- **Completion Time**: Total time from task start to functional implementation
- **Defect Count**: Static analysis via SonarQube (critical and major issues only)
- **Bug Resolution Time**: Time to fix identified defects during testing phase
- **Code Quality**: Cyclomatic complexity and maintainability index

3.3.2 Trust Calibration Survey

Administered post-task using validated scales:

1. **Reliability Trust** (α=0.89): "I trust the AI's code suggestions are correct"
2. **Competence Trust** (α=0.92): "The AI understands programming requirements well"
3. **Benevolence Trust** (α=0.85): "The AI suggestions help achieve my goals"
4. **Overall Trust** (α=0.91): Composite score across all dimensions

Scale: 1 (Strongly Disagree) to 7 (Strongly Agree)

3.4 Data Analysis

Statistical analyses included:

- Repeated measures ANOVA for efficiency metrics
- Wilcoxon signed-rank tests for non-parametric comparisons
- Effect sizes calculated using Cohen's d
- Qualitative analysis of interview transcripts using thematic coding

4. RESULTS

4.1 Task Efficiency Analysis

Table 1: Programming Task Completion Times

Group	Solo Mode (M±SD)	AI-Assisted (M±SD)	Effect Size (d)	p-value
Novice	72.4±12.3 min	58.1±9.7 min	1.29 (large)	< 0.001
Intermediate	60.3±8.9 min	49.2±7.1 min	1.38 (large)	< 0.001
Senior	48.5±6.2 min	45.0±5.8 min	0.58 (medium)	0.023

Key Finding: AI assistance produced significant efficiency gains across all experience levels, with the largest improvements observed in novice developers (19.7% reduction in completion time).

4.2 Defect Incidence Analysis

Table 2: Defect Rates Per Programming Task

Group	Solo Mode (M±SD)	AI-Assisted (M±SD)	% Change	p-value
Novice	2.1±1.4	3.6±1.9	71%	< 0.001
Intermediate	1.5±0.9	1.9±1.1	27%	0.041
Senior	0.9±0.7	1.1±0.8	22%	0.187

Critical Finding: Novice developers showed a statistically significant 71% increase in defect rates when using AI assistance, while senior developers showed no significant change.

4.3 Trust Calibration Results

Table 3: Trust Scores by Experience Level

Trust Dimension	Novice (M±SD)	Intermediate (M±SD)	Senior (M±SD)	F-statistic	p-value
Reliability	5.8±0.9	4.9±1.1	4.2±1.2	F(2,69)=15.2	< 0.001
Competence	6.1±0.8	5.2±1.0	4.5±1.1	F(2,69)=18.7	< 0.001
Benevolence	5.9±1.0	5.1±1.2	4.7±1.0	F(2,69)=8.9	< 0.001
Overall Trust	5.9±0.7	5.1±0.9	4.5±0.8	F(2,69)=20.1	< 0.001

Trust Calibration Pattern: Novice developers exhibited significantly higher trust scores across all dimensions, suggesting potential over-reliance on AI suggestions.

4.4 Qualitative Insights

Novice Developer (P-07): *"I trusted Copilot completely because it seemed to know what I needed. I didn't really question the suggestions much."*

Senior Developer (P-43): *"I used Copilot as a starting point, but I always reviewed and modified the code. It's like having a junior developer pair with you—helpful but needs supervision."*

Intermediate Developer (P-28): *"Sometimes I caught myself accepting suggestions too quickly. The efficiency gain made me less careful about reviewing the code."*

5. DISCUSSION

5.1 The Expertise Paradox

Our findings reveal a paradoxical relationship between developer expertise and AI assistance benefits. While novice developers achieve the greatest efficiency gains (19.7% time reduction), they also experience the highest increase in defect rates (71% increase). This suggests that AI copilots may create a "competency trap" for inexperienced developers.

Theoretical Framework: This pattern aligns with Dreyfus and Dreyfus's skill acquisition model, where novices rely heavily on rules and external guidance, making them more susceptible to accepting AI suggestions without critical evaluation.

5.2 Trust Miscalibration Effects

The inverse relationship between experience level and trust scores indicates significant trust miscalibration among novice users. High trust combined with limited ability to evaluate AI suggestions creates conditions for systematic error propagation.

Implications for Training: Our results suggest that AI copilot training should emphasize:

- Critical evaluation skills for novice developers

- Calibrated skepticism toward AI suggestions
- Understanding of AI limitations and failure modes

5.3 Practical Implications

For software engineering practice, our findings suggest:

1. **Experience-Dependent Deployment**: Organizations should consider experience-based guidelines for AI copilot usage
2. **Enhanced Code Review**: Teams with novice developers using AI assistance may require additional review processes
3. **Training Programs**: Structured onboarding for AI tools should emphasize critical thinking skills

5.4 Limitations

Several limitations constrain the generalizability of our findings:

- **Task Scope**: Limited to web API development tasks; results may not generalize to other programming domains
- **AI System**: Evaluation focused on GitHub Copilot; other AI assistants may exhibit different patterns
- **Duration**: Short-term study may not capture long-term adaptation effects
- **Sample**: Industry professionals may differ from academic or open-source developers

6. FUTURE WORK

Future research should investigate:

1. **Longitudinal Adaptation**: How do trust patterns and defect rates change over extended AI copilot usage?
2. **Domain Specificity**: Do our findings replicate across different programming domains (mobile, systems, etc.)?
3. **Intervention Design**: What training interventions can improve novice developers' critical evaluation skills?
4. **Explainable AI**: How do confidence scores and explanations affect trust calibration and code quality?

7. CONCLUSION

Human-AI pair programming represents a transformative shift in software development, offering substantial efficiency benefits while introducing novel challenges around trust calibration and code quality. Our empirical investigation reveals that the benefits and risks of AI assistance are not uniformly distributed across experience levels.

The finding that novice developers experience both the greatest efficiency gains and the highest defect rate increases highlights the need for experience-adapted approaches to AI copilot deployment. Organizations adopting these tools must balance productivity improvements against quality risks, particularly for less experienced team members.

Future AI copilots should incorporate mechanisms for promoting appropriate trust calibration, such as confidence indicators, explanation facilities, and adaptive suggestion strategies based on user expertise. The path forward requires not only technological advancement but also careful consideration of human factors in AI-assisted software development.

REFERENCES :

[1] M. Bansal, R. Jain, and M. Mathur, "Exploring Trust in AI-Driven Code Assistants," *Proc. of the 45th International Conf. on Software Engineering*, pp. 1012-1023, 2023.

[2] A. Ziegler and J. Müller, "Cognitive Load in Pair Programming with AI," *Journal of Empirical Software Studies*, vol. 29, no. 4, pp. 341-358, 2022.

[3] K. Lee et al., "A Study on GitHub Copilot's Impact on Code Quality," *IEEE Software*, vol. 40, no. 2, pp. 87-95, Mar. 2023.

[4] J. Smith and L. Tao, "Explainability and Reliability in Code-Generating AI," *ACM Computing Surveys*, vol. 55, no. 1, pp. 1-33, 2022.

[5] C. Nguyen and M. Thomas, "AI Tools and Developer Onboarding: A Case Study," *IEEE Trans. on Software Engineering*, vol. 49, no. 1, pp. 66-79, Jan. 2024.
[6] S. E. Dreyfus and H. L. Dreyfus, *Mind Over Machine: The Power of Human Intuition and Expertise in the Era of the Computer*, Free Press, 1986.
[7] R. C. Mayer, J. H. Davis, and F. D. Schoorman, "An Integrative Model of Organizational Trust," *Academy of Management Review*, vol. 20, no. 3, pp. 709-734, 1995.