The Impact of AI on Code and Productivity: A Case Study of GitHub Copilot Adoption in a Large Enterprise

Kajin Karunakaran[1], Chandni K Nair[2]
[1,2]*Scrum Master, Siemens Technology and Services Pvt Ltd, Bengaluru, India*

Abstract: **Artificial Intelligence (AI) tools are increasingly integrated into software development workflows, promising significant productivity gains and reshaping organizational practices. This article presents our practical experiences with GitHub Copilot (GHCP) in a large-scale enterprise environment. We highlight measurable improvements in development speed, collaboration across developer and non-developer roles, and innovations in code review and tool development, which we have experienced within our organization. At the same time, we discuss challenges encountered when applying AI-assisted coding to large-scale legacy systems. We also demonstrate how GenAI catalyzed internal innovation— supporting POCs, legacy modernization, domain specific copilots, and full-fledged AI assistants. These insights provide empirical evidence that AI not only accelerates coding but also redefines roles, workflows, and organizational learning.**

Index Terms: **Generative AI, GitHub Copilot, Impact of AI, Productivity, Software Development.**

I. INTRODUCTION

The integration of AI into software engineering has transformed how developers approach coding, debugging, and documentation. GitHub Copilot, powered by advanced language models, offers real-time code suggestions and contextual completions. While its benefits are widely acknowledged, empirical evidence from organizational adoption remains limited. This article aims to provide insights from our collective experience of deploying GHCP across diverse roles within our company, including developers, scrum masters, product owners, and architects. By documenting both successes and challenges, we contribute a case study that illustrates the broader impact of AI on productivity and code quality.

II. PRODUCTIVITY GAINS ACROSS ROLES

A. Developers

1) Accelerated Development: Developers within our organization reported a 25-40% reduction in time spent on routine coding tasks. Boilerplate code, repetitive patterns, and standard API integrations were completed significantly faster.
2) Improved Code Quality: AI-assisted suggestions encouraged adherence to best practices and reduced syntactic errors, leading to cleaner commits and fewer defects.
3) Increased Test Coverage: The introduction of AI assistance led to a substantial improvement in unit test coverage, with several teams reaching as high as 97% line coverage

What worked best:
Scaffolding tasks including controllers, adapters, serializers, test templates, and repetitive integration code, proved highly effective for AI-assisted generation, and refactoring small-to-medium modules also worked well when the relevant context fit within the IDE window and architectural boundaries were clear.

What didn't work:
Large, domain-heavy legacy modules where the relevant context spans multiple repositories or architectural layers.

B. Scrum Masters

1) Workflow Automation: Several scrum masters leveraged GHCP to develop tools that automated manual workflows, for which they earlier used to depend on Developers. Examples include dashboards pulling sprint data from JIRA and TFS for multiple scrum teams, reducing reporting overhead.

2) AI-based Planning Poker: One scrum master, with only basic Python knowledge, built an AI-based planning poker tool using organizational LLM APIs. This tool allowed teams to estimate backlog items collaboratively, with the AI agent providing historical context from JIRA and GitLab repositories. This improved the quality of estimation during the team backlog refinements and also reduced the overall refinement time by around 10%.

Why this matters:
This is not just productivity gain — it is capability expansion. GHCP enabled non developers to create internal tools on their own, reducing dependency on developers and allowing teams closest to the problems to automate their own workflows.

C. Product Owners

1) Backlog Grooming: Product owners used GHCP to script queries for backlog analysis, enabling faster prioritization and data-driven decision-making.
2) Cross-role Collaboration: AI lowered barriers for non-developers to contribute technical artifacts, fostering stronger collaboration between product and engineering teams.
3) Quick UI Mockups: Product owners used GHCP to rapidly generate low fidelity UI screens from natural language prompts, enabling faster requirement clarification and earlier stakeholder feedback without waiting for design or engineering support.
4) Epic and US Templatized Content Generation: Product owners used GHCP to generate epics and user stories in organization approved templates, accelerating requirement authoring while ensuring consistency and compliance with defined standards. This resulted in reduction in writing time of Epic and Stories from around 30mins to around 10mins, per item.

Why this matters:
These examples illustrate how AI shifts product ownership from being primarily coordination-focused to being increasingly execution-capable. By lowering the technical barrier to data analysis, workflow automation, and artifact generation, GHCP enables product owners to move faster from insight to action.

III. CODE REVIEW INNOVATIONS

A. Architect-defined Standards

Team architects codified project-specific guidelines in .md files within repositories. Developers then used GHCP to self-review code against these rules, reducing dependency on manual oversight. This reduced the need for architects to review every change and helped developers understand the reasons behind the standards in their everyday work.

B. Automated Review Tooling

A team member developed a GHCP MCP server-based tool that reviewed differential code changes directly within VS Code before merge requests. This innovation streamlined review cycles and improved consistency.

Practical takeaway:
Treat AI as a pre-review assistant that enforces standards and catches routine issues, not as the final arbiter of architectural correctness.

IV. NOVEL TOOL DEVELOPMENT BY NON-EXPERTS

Democratization of AI Tooling: Non-developers, including scrum masters and product owners, successfully built tools with GHCP despite limited programming backgrounds.

Case Study – Multi Source Project Dashboard Web Application

A scrum master developed a web-based project dashboard application that aggregated data from multiple enterprise tools, including Jira, SonarQube, GitLab, and an Open-Source Software (OSS) clearing platform. Using GHCP as a development accelerator, the scrum master integrated APIs from these systems to provide a unified, near real time view of delivery, quality, and compliance metrics.

The dashboard exposed role specific insights for diverse stakeholders—developers, product owners, quality leads, architects, and management—distributed across multiple geographical locations. Metrics included sprint progress, defect trends, code quality indicators, pipeline health, OSS compliance

status, and release readiness signals, significantly reducing reliance on manual status reports and spreadsheets.

Why this matters:
This example highlights practical organizational leverage, not just coding speed. GHCP enabled a non-developer role to quickly translate recurring reporting pain points into a scalable, shared artifact that improved transparency, alignment, and decision making across distributed teams—without adding load to core engineering backlogs.

V. THE HARD PART: LEGACY CODE AND CONTEXT LIMITS

Our biggest friction points appeared when applying GHCP to large legacy systems. In our environment, many business-critical projects are written in C++ and have evolved over ~20 years, accumulating architectural drift, partial refactors, and mixed idioms (STL, custom allocators, legacy frameworks). In these codebases, developers consistently reported lower accuracy and higher variance in GitHub Copilot (GHCP) suggestions compared to modern, modular repositories. The root causes clustered around context fragmentation, implicit domain knowledge, and pattern inconsistencies.

1) Contextual limitations: GHCP struggled with complex, domain-specific systems when relevant code and requirements exceeded the manageable scope of what the model could "see".
2) Inconsistent suggestions: Completions occasionally conflicted with established architectural patterns, requiring careful developer validation.
3) Maintenance overhead: Integrating AI suggestions into legacy systems sometimes increased short-term effort due to refactoring and validation needs.

GHCP adoption in legacy environments revealed structural and skill-level constraints, as developers accustomed to monolithic systems struggled to apply AI assistance effectively without clearer boundaries, documentation, and modernization practices.

Interpretation:
AI cannot remove the complexity in legacy systems — it only makes it visible sooner. Productivity still depends on clear system boundaries, good documentation, and careful human review.

VI. ACCELERATED LEARNING AND PROOF-OF-CONCEPTS

A. Rapid Prototyping

We noticed that developers have successfully utilized GHCP to bootstrap projects in unfamiliar technologies, including cloud-native frameworks and AI/ML libraries. This capability has enabled faster project initiation across diverse tech stacks, regardless of prior team expertise.

B. Reduced Learning Curve

GHCP's contextual code suggestions have directly accelerated experimentation and skill development. The time required for our developers to gain working proficiency in new domains has been measurably reduced, transforming the traditional learning curve into a more streamlined experience.

C. Innovation Velocity

Proof-of-concepts (POCs) development timelines have shown significant improvement with GHCP integration. Same old teams can now validate ideas and evaluate emerging technologies with substantially less upfront time investment, enabling more agile decision-making around technology adoption and innovation strategies.

VII. HOW TO MAKE AI SAFER AND MORE RELIABLE

A. Human Oversight is Critical

We found that GHCP works best when teams operationalize the principle: "AI suggests; humans decide." This includes checking architectural fit, validating edge cases, and running security-relevant reviews as usual.

B. Incremental Adoption

GHCP proved most effective when introduced gradually, starting with modular components rather than monolithic legacy systems.

C. Training and Guidelines

Establishing internal best practices for AI-assisted coding improved consistency and reduced misuse.

D. Cross-role Empowerment

AI tools democratized coding across roles, but governance and oversight remained essential to prevent misuse.

VIII. PRACTICAL ADOPTION PLAYBOOK (FOR PRACTITIONERS)

Below is the compact playbook we recommend to teams adopting AI-assisted coding tools.

A. Start where AI is Strongest

Begin with boilerplate-heavy work.

Add automation scripts for workflows close to the role's pain points (dashboards, reporting, backlog analytics).

B. Codify "how we write code here"

Capture architectural and coding standards in repo markdown.

Encourage "AI-assisted self-review" against these rules before merge request creation.

C. Use AI to reduce review load, not to remove review

Apply AI to differential pre-review (IDE-integrated checks), then keep humans for final correctness and architectural alignment.

D. Don't ignore legacy, reframe it

For legacy systems, treat AI as a navigation and refactoring assistant for small slices, and invest in better documentation to improve context.

IX. FUTURE DIRECTIONS

Our experience suggests that AI tools like GHCP are best positioned as accelerators rather than replacements for human expertise. Future research should explore:

1) Contextual Awareness: Techniques for enhancing AI contextual understanding in legacy systems.
2) Productivity Metrics: Long-term metrics for quantifying productivity gains beyond initial adoption.
3) Ethical Considerations: Ownership of AI-generated code, bias in suggestions, and organizational implications of widespread AI integration.
4) Expanding AI Literacy: Building AI literacy across non-developer roles to maximize organizational impact.

X. CONCLUSION

This case study demonstrates that AI-assisted coding tools, when introduced thoughtfully, can generate meaningful productivity and capability gains across a wide range of roles within a large enterprise. GitHub Copilot proved most effective as an accelerator for routine and boilerplate-heavy development, rapid prototyping, and cross-role collaboration, enabling developers to reduce time spent on repetitive tasks while improving code consistency and adherence to best practices. Beyond traditional software engineering roles, scrum masters and product owners successfully leveraged AI assistance to automate workflows, build internal tools, and generate technical artifacts—illustrating a shift from role-specific productivity improvements toward broader organizational capability expansion.

However, this study also highlights clear limitations. AI assistance struggled in large, domain-heavy legacy systems where architectural complexity, weak documentation, and fragmented context constrained model effectiveness. In these environments, AI did not eliminate complexity but instead surfaced it earlier, reinforcing the importance of modular design, clear system boundaries, and human judgment. Without strong governance, documentation, and incremental adoption strategies, AI-generated suggestions risk inconsistency and short-term integration overhead.

Overall, our experience reinforces that AI tools such as GitHub Copilot are most valuable when positioned as collaborators rather than autonomous agents—augmenting human expertise instead of replacing it.

ACKNOWLEDGMENT

We thank the employees of Discrete Automation Systems & Software department at Siemens Technology & Services Pvt. Ltd. (Bangalore) for their enthusiastic adoption of Artificial Intelligence tools such as GitHub Copilot in their daily workflows.

We also extend our gratitude to GitHub Copilot for its assistance in refining the content of this article.