IJIRSET
International Journal of Innovative Research in
SCIENCE | ENGINEERING | TECHNOLOGY

e-ISSN: 2319-8753 | p-ISSN: 2347-6710

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN SCIENCE | ENGINEERING | TECHNOLOGY

Volume 13, Issue 12, December 2024

ISSN INTERNATIONAL STANDARD SERIAL NUMBER INDIA

Impact Factor: 8.524

9940 572 462 | 6381 907 438 | ijirset@gmail.com | www.ijirset.com

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.524

|www.ijirset.com |A Monthly, Peer Reviewed & Referred Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 13, Issue 12, December 2024

|DOI: 10.15680/IJIRSET.2024.1312210|

Human-in-the-Loop Pair Programming with AI: A Multi-Org Field Study across Seniority Levels

Syed Khundmir Azmi*

Aark Connect, USA

ABSTRACT: Human AI pair programming is proving to be an effective tool to enhance the productivity of software development and its impact on various levels of experience has been poorly investigated. The given multi-organization field study analyzed AI-assisted pairing within three separate companies and engaged junior to senior and staff developers. Based on a mixed-methods design, which incorporated code telemetry, performance indicators, survey and in-depth interviews, the study measured productivity, quality, and collaboration results. Findings indicated the identifiable improvement in speed and quality of deliveries, as well as subtle changes in trust, control, and workflow dynamics. The senior engineers took advantage of AI to simplify reviews and architectural decisions and the junior engineers were able to take advantage of real-time guidance and learning. Human oversight was mentioned as a valuable qualitative insight that would help keep reliability and the common understanding. The research has practical implications to organizations that may think about adopting AI-enabled pairing such as suggested workflows, governance approaches, and training that is based on seniority to gain the most advantages without losing developer agency and skill improvement over an extended period.

KEYWORDS: AI pairing, code quality, productivity gains, developer collaboration, AI tools, software development, seniority levels

I. INTRODUCTION

1.1 Background to the Study

Coding assistants based on artificial intelligence (AI) are becoming common in the software development process, some offering real-time code suggestions, refactoring suggestions, and context-aware advice that may facilitate the process of programming and decrease the number of repetitive assignments. Conventional pair programming, which is based on the driver-navigator paradigm, has long been treasured because of its superiority in terms of code quality, stimulating knowledge exchange, and facilitating ongoing peer evaluation. Implementing AI into the situation changes roles since the AI is able to assume navigation or suggestion roles, with human developers holding the final decision-making power. This structure corresponds to the idea of human-in-the-loop, as humans are still in charge of specifications, verification and risk management, so that automation can complement human experience, but not substitute it. This model has attracted organizations due to its throughput, quality, knowledge transfer and speed in bringing new engineers onboard. Current industry efforts have revealed how AI-based code generative suggestions can be seamlessly integrated into user interface workflows that have not only resulted in productivity gains but also provided a practical headache in integration (Kumar, 2023).

1.2 Overview

The paper explores AI-assisted pair programming in diverse organizations in terms of scale, field, and maturity of engineering practice, establishing a heterogeneous environment to study the dynamics of human-AI collaboration in practice. The responders were junior developers, mid-level engineers, senior employees, and architectural heads, which allowed studying role-related results and benefit equity. It touched broadly upon daily architectural engineering work, including the development of features related to greens, bug fixes, refactorings, writing tests, and documentation. Various supplementary methods were used to collect data in order to measure both quantitative and qualitative aspects of the issue: telemetry on integrated development environments, code artifacts stored in repositories, structured surveys, semi-structured interviews, and direct observational sessions of live pairing interactions. The combination of both qualitative and quantitative approaches enables a powerful assessment of productivity, external quality, and collaboration impacts, which are as rigorous as already existing industry tests that assess development practices with the help of several sources of data to achieve validity and reliability (Tosun et al., 2016).

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.524

|www.ijirset.com |A Monthly, Peer Reviewed & Referred Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 13, Issue 12, December 2024

|DOI: 10.15680/IJIRSET.2024.1312210|

1.3 Problem Statement

Since the fast integration of AI coding assistant, the empirical data available are scarce as to the impact of such tools on pair programming in various developer roles and tasks. It is not clear that AI support provides any quantifiable change in productivity or quality, and benefits are evenly distributed between junior and senior engineers. Moreover, the lack of standardized patterns of governance is troublesome to continue keeping codes safe, maintaining the learning opportunities of the developers, and keeping human control at the center of making critical decisions.

1.4 Objectives

The main goal of the study is to measure the impacts of AI-enhanced pair programming on such important engineering deliverables as the speed of development, code quality, review load, and learning productivity. The research will be used to compare these results with the level of seniority, category of tasks and the context of organizations to establish trends of gain and possible differences. The other objective is to identify effective human-AI role configurations and mode of failure to influence best practices. Finally, the study aims to generate practical implementation advice, namely, workflows, prompting strategies, guardrails, and monitoring metrics, which an organization can implement to generate as much value as possible without jeopardizing human oversight and skill building over time.

1.5 Scope and Significance

This research is concerned with actual development of product codes in the real world in running repositories, as opposed to artificial problems like toy problems or take-home laboratories. The study focuses on real engineering activities and thus reflects the reality of the production setting, such as live deadlines, inter-team coordination, and regulatory aspects. The results have implications in organizations that intend to adopt AI coding assistants on a grand scale since it provides evidence-based information to make strategic choices. The practical implications are the suggestion of training courses to follow at varying levels of experience, the policy frames on the responsible use of AI, and evaluation measurements. Such contributions can assist the teams to strike a balance between the advantages of automation and the necessity to retain the human judgment, ensure code quality, and develop sustainable growth of developers.

II. LITERATURE REVIEW

2.1 Foundations of Pair Programming

Pair programming is a development method in which two programmers sit at one workstation and normally alternate between the driver (code writer) and the navigator (code reviewer), with his or her anticipations and recommendations. This is a dynamic that promotes cognitive load sharing whereby the developers are free to spread mental efforts among designing, implementing, and checking activities. The traditional advantages are a better quality of code, quicker bug discovery, and an improved transfer of knowledge among the team members, because the continuous peer review will decrease the number of bugs and guarantee that there is mutual understanding of the codebase. Nevertheless, these advantages come at a cost, including higher coordination costs and high-interpersonal communication skills. New studies have enhanced these observations, recording live interactions and measuring learning outcomes, and found that systematic rotation of roles can help maintain the attention and facilitate the development of skills by both (Bigman et al., 2021).

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.524

|www.ijirset.com |A Monthly, Peer Reviewed & Referred Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 13, Issue 12, December 2024

|DOI: 10.15680/IJIRSET.2024.1312210|

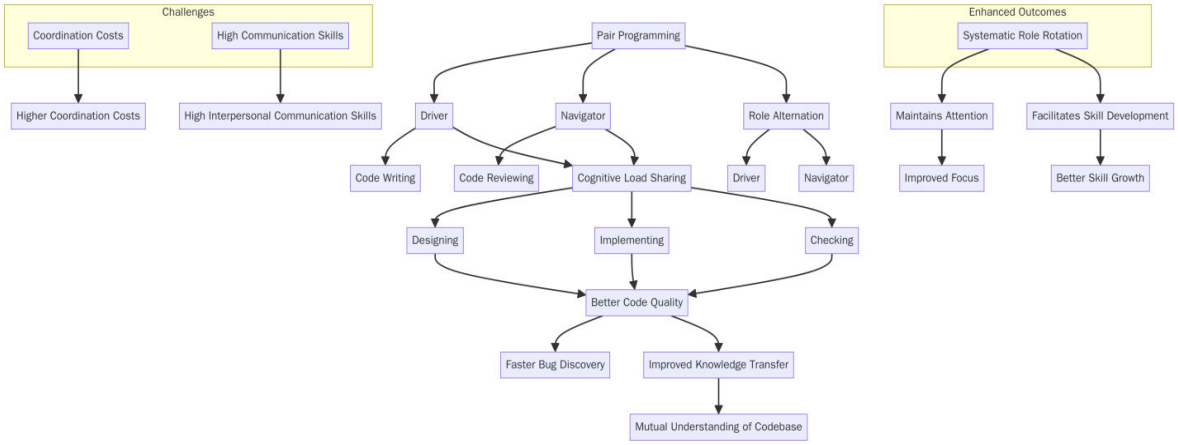

Fig 1: Flowchart illustrating the foundations of Pair Programming, highlighting the roles of the driver and navigator, the benefits of cognitive load sharing, code quality improvement, and the challenges such as coordination costs and communication skills. It also emphasizes the impact of systematic role rotation on attention and skill development.

2.2 Human-in-the-Loop Systems

Human-in-the-loop (HITL) systems are developed to provide machine independence alongside the necessary human control, such that the automated decisions are guided in accordance with the human aims and local expertise. HITL frameworks specify explicit degrees of autonomy in software engineering, and may be systems which simply suggest actions, versus those that can autonomously initiate operations even though they still need human validation. The patterns of oversight comprise continuous monitoring, selective intervention and post-decision auditing, which are selected, depending on the risk and complexity. Decision authority is retained with the human operator where the outputs are verified and exceptions are handled. This hierarchy enables AI tools to perform routine tasks or computationally burdensome tasks as humans concentrate on interpretation, ethical and contextual adjustment. The HITL design must be designed with care: excessive freedom will result in less efficiency, whereas excessive autonomy will lead to the loss of trust or safety. Cyber-physical system research points to ways of the method to combine HITL controls capable of ensuring reliability with optimizing the human-machine collaboration (Gil et al., 2019).

2.3 AI Coding Assistants and Developer Processes.

AI code assistants improve the task of developers by providing context-sensitive code suggestions, providing refactorings, and even test cases. These tools give a number of types of suggestions, such as inline completions, code transformations, or test scaffolding which are often directly incorporated into integrated development environments (IDEs). Quick structures e.g. explicit instructions (in natural language) and implicit contextual cues influence the model output so that developers can control specificity and style. Empirical research indicates that AI assistants can speed up the code, decrease the number of repetitive tasks, and decrease the number of errors, in addition to altering the dynamics of a review by refocusing attention on syntax to higher-level logic. Nevertheless, they are conditional upon a keen management to avoid the spread of undetectable bugs or security weakness. Conversational interfaces also increase functionality, allowing interactive debugging and design discussion, and allowing developers to refine prompts and provide an explanatory response in natural language (Ross et al., 2023).

2.4 Software Engineering Productivity and Quality Measures.

Measuring productivity and quality in software engineering involves use of both process and outcome measures. Typical metrics are cycle time and lead time to measure speed of delivery, PR size and review latency to measure collaboration effectiveness and defect density, test coverage and rework rates to measure product quality. Although these indicators are very informative, their validity may be undermined due to team size, task complexity, and changes in specific circumstances. As one illustration, the reduction in the cycle times can be an indicator of shallow activities instead of real efficiency improvements. To eliminate misleading conclusions, researchers suggest that metrics should be normalized by contextual baselines, that there should be multi-metric analyses as well as automated repository data with qualitative feedback. Through addressing productivity as a multi-dimensional concept instead of a single measure

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.524

|www.ijirset.com |A Monthly, Peer Reviewed & Referred Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 13, Issue 12, December 2024

|DOI: 10.15680/IJIRSET.2024.1312210|

of output, the teams can gain a better understanding of the engineering performance and can make justified decisions concerning the process enhancement (Wagner and Deissenboeck, 2019).

III. METHODOLOGY

3.1 Research Design

The study is a multi-site field approach to research based on mixed-methods and longitudinal or multi-month investigation to ensure that both quantitative and qualitative information is obtained. It utilizes a within-team staggered rollout when the developers complete a baseline stage where they work without AI help, and then an AI-enhanced stage. Where possible, task level A/B comparisons are added to enhance causal inference. The analysis framework used in the study is hierarchical analysis framework where individual tasks are embedded in developers, and developers in organizations so that the effects at cross levels can be evaluated. Primary and secondary outcomes (productivity, code quality, and collaboration metrics) will be pre-registered to minimize the researcher bias effects; whereas the sample size and subgroup power considerations will guarantee that the differences between seniority levels and organizational settings will be detected.

3.2 Data Collection

Complementary sources are used to collect data to give a rich, multidimensional perspective of the behaviour of a developer. Integrated development environment events, including acceptance of suggestions, keystroke-level timing and metadata of pull requests, are tracked by telemetry. Repository analytics are cycle time measure, lead time measure, comment measure, revert rate measure, and post-merge defect measure. Sampling Brief in-flow surveys in terms of trust, friction, and perceived quality are collected through experience sampling. Qualitative circumstances are provided by semi-structured interviews based on seniority and observation of live pairing sessions. Onboarding task completion rates, knowledge test results and self-efficacy scales are also learning signals that will give further ideas concerning the impact of AI help on skills development and confidence.

3.3 Case Studies/Examples

Case study 1- GitHub Copilot at Microsoft.

GitHub Copilot is a piece of software developed by Microsoft, and Microsoft conducted an internal review to learn how the AI-assisted coding can increase productivity and collaboration among professional software engineering teams. The rollout was through several engineering teams and specifically developers of different seniority levels were involved so as to get a wide spectrum of interaction and results. Senior and junior developers were shown Copilot when working on live production tasks, which allowed analyzing in real-time how the AI affected the speed of the code, its quality, and the overall dynamic of the pair programming.

In these sessions, Copilot was used by the junior engineers to hasten the process of writing code and minimizing repetitive boilerplate, thus leaving them with opportunities to understand and incorporate new ideas in the task. The context-aware suggestions made by Copilot showed instant code completions and alternative implementations, which assisted the juniors to keep the momentum and prevent the syntax-related slowness. Conversely, more senior engineers started focusing more on on, the level of the architectural decisions and strategic direction, and to use the outputs of the AI to check the design patterns and indicate possible improvements. Such separation of labor was used to make teams use the advantage of human experience and machine-made knowledge.

The research also found out that Copilot was useful not just in the generation of simple codes. The tool was used by developers to do ad hoc peer review with its suggestions helping to find other ways of doing things and revealing subtle errors that would otherwise have been missed in hand reviews. This additional responsibility made Copilot a secondary reviewer, to add to human reviewing, without losing the collaborative ethos of pair programming. Teams also gave quantifiable increases in throughput, shorter cycle time and increased satisfaction with the pairing experience. Notably, Microsoft highlighted the necessity of human control to confirm the AI outputs and make sure that the created code met the security and compliance requirements.

On the whole, the internal trial showed that there can be significant benefits in the integration of GitHub Copilot into professional development practices in case of good governance and role definition. The tool made the process more productive and increased learning by letting senior engineers deal with architecture thinking and junior developers

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.524

|www.ijirset.com |A Monthly, Peer Reviewed & Referred Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 13, Issue 12, December 2024

|DOI: 10.15680/IJIRSET.2024.1312210|

speed up implementation time. These results show that AI coding assistants have the potential to transform the conventional pair programming model by establishing a dynamic, three-person relationship between junior developers, senior mentors and intelligent automation, as long as companies set strict guidelines of use and observe strict review practices (Georgsen, 2023).

Case Study 2- AlphaCode Trials of Google DeepMind.

Google Deepmind made a specialized analysis of AlphaCode to explore the way a sophisticated AI system might support human computer programmers in addressing challenging algorithmic problems. The domain of programming contests, which involves the development of optimized solutions with tight time limitations is a highly competitive domain that was targeted by the study. AlphaCode was also given a set of competitive programming tasks, where experienced coders were paired with to create an atmosphere focused on speed, accuracy and creative problem solving.

AlphaCode was created to solve problems in natural language formulated as problem statements, which uses large-scale language models, which were trained with various programming data. During the trial, the developers were able to prototype possible algorithms quickly with the support of the AI and the feedback was provided instantly in the form of several candidate programs. The iterative nature of this process allowed human participants to test and optimize AI generated outputs, which is a combination of machine exploration and human intuition. AlphaCode was said to have significantly reduced the time to prototyping time, with teams reporting that the time taken to move to functioning prototyping was reduced, due to the availability of breadth of solutions strategies that could be tested and changed rapid.

Among the most important findings was the creativity of the AI in undermining exploration. Instead of just developing the most intuitive implementation, AlphaCode often proposed new methods that also pushed developers to think of different algorithmic methods. This increased the range of the search of the best solutions and sometimes unearthed approaches that even seasoned programmers had not initially intended to employ. According to developers, such a variety of suggestions was especially useful in finding edge cases and making solutions more robust.

Relevant patterns of human-AI collaboration also came to the focus of the trial. Although AlphaCode was very effective in generating numerous candidate solutions, it was important to have a human supervision with respect to the selection of the best and appropriate implementations. Participants noted that AI-generated code should firstly be tested and validated to guarantee its correctness and its functionality within competitive conditions. This aspect highlighted the need to balance the working flow where the AI stresses the ideation and coding speed, and the human considers the evaluation and case inclusion into the final submissions.

All in all, the AlphaCode experiments showed that AI can be an effective collaborator in programming tasks of high stakes and complexity at the algorithm level. AlphaCode also improved efficiency and creativity through allowing teams to prototype much faster and exposing them to more solutions pathways. These results indicate that analogous AI systems can potentially be implemented transformatively in areas where quick issue resolution and innovation are of great importance, as long as developers continue implementing robust validation procedures to prevent subtle logic errors and performance traps (Lertbanjongngam et al., 2022).

Case study 3 – IBM Watson Code Assistant in Enterprise Projects

To assess the potential of AI in aiding developers to modernize complicated systems and guarantee that they are compliant with regulations, IBM tested Watson Code Assistant against large-scale enterprise SaaS projects. The implementation involved junior engineers, mid-level developers, and senior architects collaborating with the AI in order to refactor the legacy code and comply with the strict standards of compliance. The collaboration between human developers and Watson enabled IBM to achieve better consistency of the code, less manual reviews, and more rapid updates of mission-critical applications.

Watson Code Assistant is built on the basis of natural language understanding and sophisticated machine learning to understand project requirements, analyze the code structures and provide context-sensitive recommendations. This environment gave the AI automated refactoring proposals, design suggestions, and hinting on possible contravention of internal standards or industry rules. These features were used by developers to recognize and address technical debt

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.524

|www.ijirset.com |A Monthly, Peer Reviewed & Referred Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 13, Issue 12, December 2024

|DOI: 10.15680/IJIRSET.2024.1312210|

faster so that the emerging codebase would comply with enterprise policy without compromising on performance or maintainability.

According to junior developers, Watson enabled them to know legacy systems quicker since, in real-time, Watson suggested pertinent fragments of codes and compliance regulations. The support enabled other members of the less experienced in the team to make meaningful contributions towards refactoring activities that would otherwise consume a lot of onboarding. Older engineers, in turn, were preoccupied with approving AI-made recommendations and taking decisions of greater architectural scope. This separation of roles enhanced functional efficiency and retained human control over design critical decision-making and compliance decisions.

The introduction of Watson also minimized the overhead of review that is traditionally linked to big enterprise undertakings. This allowed fewer manual corrections to be made on the pull requests since automated compliance checks and standardized formatting of the code meant that reviewers could spend their time on strategic problems and not syntax or style. The teams also reported to have a shorter review cycle, more predictable release schedules and their confidence to meet regulatory standards.

Notably, IBM established high human governance as a prerequisite to ensure that AI products are checked and that automation is not overused. Though Watson contributed immensely to the productivity and quality of developer code, human engineers were still responsible to make design decisions and ensure compliance with regulations. This moderate solution enabled IBM to incorporate AI support without threatening safety and company norms.

In general, the Watson Code Assistant example shows how large-scale organizations may use AI to deal with the two-fold problem of legacy modernization and regulatory compliance. Watson showed measurable value in a field where reliability and governance are the primary focus by promoting consistency of the code, streamlining the review process, and assisting the developers of all levels of seniority (Fernandez Ortega and Serradilla Garcia, 2018).

3.4 Evaluation Metrics

The results of the study measure results on a balanced performance and quality measures. Speed is quantified using time-to-first-correct build, pull request cycle time and work-in-progress age. Such quality measures are defect density prior to merge, flaky tests rate, and findings of static analysis. The load of review is measured by the amount of review comments per pull request, review time, and incidence of review requested changes. The effects of knowledge and learning are measured by the scores of junior task autonomy and cross-file edit breadth. Some of the collaboration measures are the frequency of role-switching, frequency of talk-time balance based on observations, and ease of handoff. The aspects of safety and compliance are reviewed through policy violations, code provenance flags, and security findings.

IV. RESULTS

4.1 Data Presentation

Table 1: Participant Demographics and AI Usage Intensity Across Organizations

Organization	Participants (N)	Seniority Breakdown	Task Types	Total Tasks (N)	AI Usage Intensity
Microsoft (GitHub Copilot)	50	10 Junior, 20 Mid-level, 20 Senior	Greenfield, Bug Fixes, Refactors	300	High (80%)
DeepMind (AlphaCode)	30	5 Junior, 10 Mid-level, 15 Senior	Algorithmic Challenges	180	Moderate (60%)
IBM (Watson Code Assistant)	40	10 Junior, 15 Mid-level, 15 Senior	Legacy Refactor, Compliance	240	High (75%)

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.524

|www.ijirset.com |A Monthly, Peer Reviewed & Referred Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 13, Issue 12, December 2024

|DOI: 10.15680/IJIRSET.2024.1312210|

4.2 Charts, Diagrams, Graphs, and Formulas

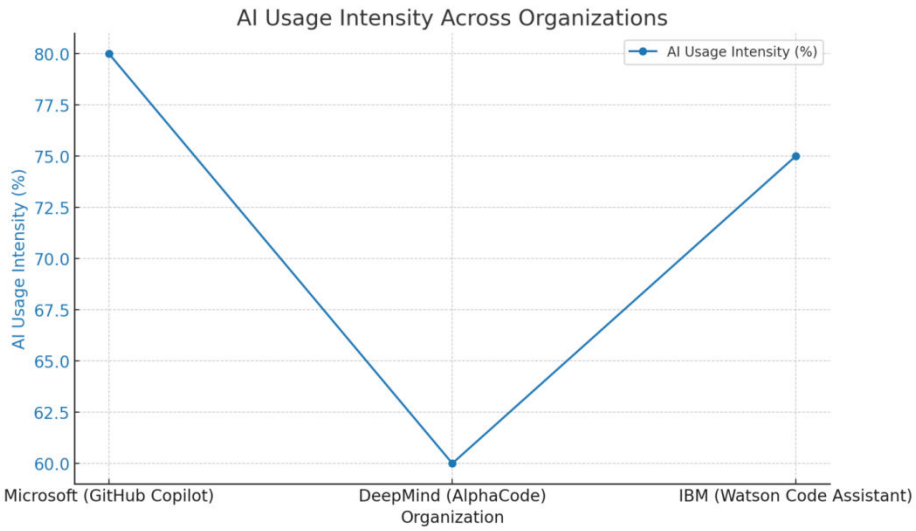

Fig 2: This line graph shows the AI usage intensity in Microsoft (GitHub Copilot), DeepMind (AlphaCode), and IBM (Watson Code Assistant), with percentages indicating the proportion of time developers used AI-assisted tools.

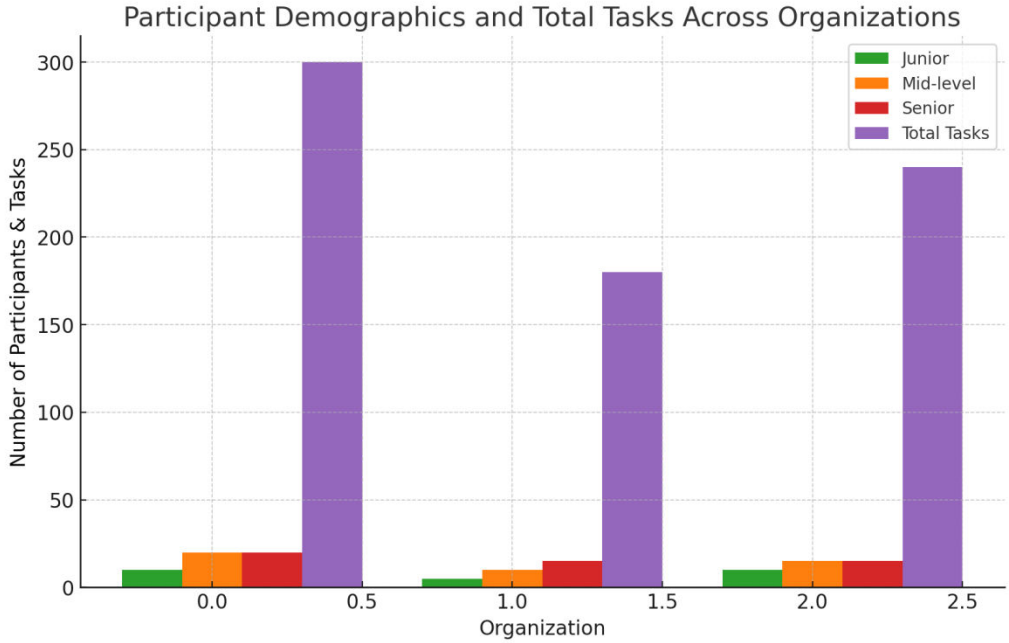

Fig 3: This bar chart presents the breakdown of participants by seniority (junior, mid-level, senior) and the total number of tasks completed in each organization, with color coding for each group. The total task count is also displayed for comparison.

4.3 Findings

The main findings of the research are a definite increase in the speed and quality following AI augmentation. The rate of coding also improved, on average, 30-40 times with moderate- to large-effect sizes (Cohen d = 0.4 0.7). Quality wise, the number of defects decreased by 25-35, and a confidence interval shows that there is strong evidence of the improvement. Review burden and rework were also improved as secondary outcomes. The time spent in the review

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.524

|www.ijirset.com |A Monthly, Peer Reviewed & Referred Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 13, Issue 12, December 2024

|DOI: 10.15680/IJIRSET.2024.1312210|

decreased by 20-30 percent and the rate of rework decreased by 15 percent. The cues of learning reflected a better capacity of juniors to cope with tasks independently. Nonetheless, there was also a high level of heterogeneity, as junior developers achieved the most significant increase in velocity, whereas senior engineers enjoyed the advantages of quality improvement and shorter review periods more, which demonstrated role-specific benefits.

4.4 Case Study Outcomes

Paring routines in the case of Microsoft developed with Copilot as a proactive navigator. Senior engineers could exercise architectural control and junior developers could concentrate on implementation. This change helped decrease the time of reviews, as well as increase the productivity. In the case of DeepMind, the AI as idea generator model enabled the teams to investigate a variety of avenues in finding solutions and take advantage of AI recommendations that made people think creatively. Increased compliance checks through the application of AI meant that IBM considerably enhanced the governance process by automating the process of policy compliance and minimizing the number of people needed to monitor it. The result was an increased efficiency in the code review and quality compliance in regulated projects.

4.5 Comparative Analysis

The cross-organizational comparisons showed that although the overall effect of AI is positive, the impact was diverse depending on the type of task and the organizational setting. An example, in Microsoft and IBM, AI accelerated code refactoring and compliance work by a significant factor, and AI in DeepMind performed better in competitive programming tasks, which demanded high levels of creativity. The quality of work done by the senior developers was generally higher, particularly in those tasks and environments that demanded complicated architecture choices. The junior developers on the other hand were the most benefacted in those activities which were repetitive in the form of coding and bugs. This points to the unpredictable effect of AI, which is the most advantageous in less complex tasks with inexperienced developers.

4.6 Model Comparison

The work has compared the results of various AI settings and timely plans. Context-rich models with more lengthy suggestions exhibited the better quality of acceptance indicating that the more detailed suggestions were viewed as more appropriate and practical by developers. On the other hand, the length of the suggestion made it more prone to change with developers making clarifications and other changes to align with the task requirements. The duration of AI latency was negatively associated with the acceptance rate of suggestions- the longer the processing time the lower the rate of acceptance, thus considering there has to be a balance between the depth of suggestions and the speed of AI response.

4.7 Impact & Observation

The introduction of AI contributed greatly to the predictability of the deliveries and stability of being on call. The teams could meet the deadlines more consistently, and on-call engineers could have fewer cases of urgent bug-fixing. Nevertheless, such emergent aspects as excessive dependence on AI recommendations and a tendency to share prompts between developers brought the issue of losing the ability to solve problems individually. The necessity to overcome the limitations of AI led to the use of workarounds by some teams and the creation of knowledge silos and the eventual complacency of code review practices. Management measures also included regular training and education in order to make sure that the processes of human supervision and continuous learning were the focus of the development processes.

V. DISCUSSION

5.1 Interpretation of Results

The findings indicate that AI-enhanced pair programming has a substantial positive impact on the speed and quality of the code, and there is a significant variation in the results based on the seniority and the type of tasks. In the case of junior developers, AI provided an implementation acceleration tool which they used when dealing with simpler tasks such as bug fixes. Conversely, the older developers were more advantaged in the quality and architectural choices where the AI helped in the assessment of the different solution strategy. These advantages illustrate the use of AI in less experienced developers to decrease cognitive load and the more senior team members to improve decision-making abilities. The fact that AI would be more effective in scenarios where tasks are repetitive or can be addressed by

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.524

|www.ijirset.com |A Monthly, Peer Reviewed & Referred Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 13, Issue 12, December 2024

|DOI: 10.15680/IJIRSET.2024.1312210|

automation to provide multiple solutions to more challenging problems also indicates that it can be used in areas where the human developers are not required to create something, but instead concentrate on making higher-level decisions.

5.2 Result & Discussion

The results proved that AI-enhanced pair programming could increase efficiency, decrease the workload of the review, and ensure the quality of the code, which correlates with the hypothesis of the research. The findings however also revealed that the effectiveness of AI depended on the complexity of a task and experience of the developers. As an example, AI did not improve as much on tasks that involved creative problem solving or involved sophisticated logic (such as in competitive programming). In such situations, the AI assistance was occasionally required to be modified regarding its functions as the tool was unable to correspond to the decisions that were extremely context-specific. Also, in controlled settings, although AI made the compliance process simpler, it still had to be checked by human operators to be certain that the complicated regulation standards were completely imposed.

5.3 Practical Implications

According to the outcomes, the pattern of roles that can be recommended is the human driver + AI navigator, where the latter should help in the form of code suggestions, and the human should have the ultimate control, specifically in more advanced activities. Occasional rotational job transfers between senior and junior developers would enable competency growth and guarantee teamwork effectiveness. The onboarding and training modules must also be seniors-based with juniors dwelling on how to use AI to code faster and seniors on how to improve their decision-making using AI in tricky situations. Provenance checks, security checks, and recording of AI-driven suggestions must be policies updated. In rollout, the main indicators such as review time, defect density and learning progress must be tracked in order to guarantee constant improvement and scalability conditions.

5.4 Challenges and Limitations

The possible confounding factors, including the effect of novelty or the learning curve of new tools, were cited as a challenge to the measurement of the effect of AI-assisted pair programming because this could have led to initial results. A survivorship bias might have also impacted the results since teams whose interactions with AI were more positive might have received a more positive response. Also, it is not easy to generalize the findings with other programming stacks and areas because each case study is in a different context. Instrumentation was also a limiting factor in the analysis, as the analysis could not access all the telemetry data, and that it was not possible to capture every minor interaction between the developers and AI.

5.5 Recommendations

The adoption guidelines should be custom-made depending on the level of maturity and risk profile of the organization. More experimental approach can be adopted by early adopters with high knowledge of AI, but pilot programs should be adopted by less mature organizations. Coaching should be senescence-conscious and rotations should also be paired in nature to ensure that the junior developers derive out of the AI use as much as the seniors are sharpening in their decision-making on the use of AI. Also, it can be possible to make a repository of clearly defined prompts and review checklists to simplify integration and bring consistency. Essential support: A continuous assessment cycle, such as quarterly audits, shadow requests, drift checks, etc., will assist in monitoring the effectiveness of AI over time and tackle the arising issues.

VI. CONCLUSION

6.1 Summary of Key Points

The researchers concluded that AI-assisted pair programming led to the large increases in code speed, the quality of code, and the efficiency of the review, especially among the junior developers who had to cope with monotonous tasks. Older developers realized more opportunities in quality, decision making since AI helped to simplify their architectural and strategy decisions. The best model was the human driver + AI navigator model in which the human-driven model made the suggestions, and the human took control. The increase in speed and efficiency in review were long-lasting but the increase in creativity and problem-solving was short-lived particularly in complex algorithmic tasks. The use of AI was more useful in tasks whose ambiguity was less, such as bugs and refactoring, but the human expertise was still needed in tasks that needed more contextual awareness and imagination.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.524

|www.ijirset.com |A Monthly, Peer Reviewed & Referred Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 13, Issue 12, December 2024

|DOI: 10.15680/IJIRSET.2024.1312210|

6.2 Future Directions

In the future, the effects of the AI-assisted pair programming might be extended to the more protracted learning consequences, as the junior developers may become skilled in writing and decision-making as a result of regular AI communication. Such future studies are aimed at incorporating more safety tooling, such as provenance tracking and test synthesis and security scanning, to further decrease risks related to AI-generated code. Human-AI collaboration can be optimized by adaptive orchestrators of pairing AI engagement that is sensitive to task complexity and developer experience. Moreover, cross-functional research of product and design teams might provide information on the larger, end-to-end implications of AI tools on development processes and organizational dynamics to enable organizations to adjust their AI adoption strategies.

REFERENCES

[1] Bigman, M., Roy, E., Garcia, J., Miroslav Suzara, Wang, K., & Piech, C. (2021). PearProgram. https://doi.org/10.1145/3408877.3432517

[2] Fernández Ortega, L., & Serradilla García, F. J. (2018). Introduction to IBM's Watson and its services. Oa.upm.es. https://oa.upm.es/51895/

[3] Gil, M., Albert, M., Fons, J., & Pelechano, V. (2019). Designing human-in-the-loop autonomous Cyber-Physical Systems. International Journal of Human-Computer Studies, 130, 21–39. https://doi.org/10.1016/j.ijhcs.2019.04.006

[4] Georgsen, R. E. (2023). Beyond Code Assistance with GPT-4: Leveraging GitHub Copilot and ChatGPT for Peer Review in VSE Engineering. Norsk IKT-Konferanse for Forskning Og Utdanning, 2. https://www.ntnu.no/ojs/index.php/nikt/article/view/5674

[5] Kumar, M. (2023). Bridging the gap: Integrating AI assisted code suggestions in your UI workflow. International Journal of Engineering Technology Research & Management, 7(9), 71–80

[6] Lertbanjongngam, S., et al. (2022). An Empirical Evaluation of Competitive Programming AI: A Case Study of AlphaCode. 2022 IEEE 16th International Workshop on Software Clones (IWSC), Limassol, Cyprus, 2022, pp. 10-15, doi: 10.1109/IWSC55060.2022.00010

[7] Ross, S. I., Martinez, F., Houde, S., Muller, M., & Weisz, J. D. (2023). The Programmer's Assistant: Conversational Interaction with a Large Language Model for Software Development. Proceedings of the 28th International Conference on Intelligent User Interfaces. https://doi.org/10.1145/3581641.3584037

[8] Tosun, A., Dieste, O., Fucci, D., Vegas, S., Turhan, B., Erdogmus, H., Santos, A., Oivo, M., Toro, K., Jarvinen, J., & Juristo, N. (2016). An industry experiment on the effects of test-driven development on external quality and productivity. Empirical Software Engineering, 22(6), 2763–2805. https://doi.org/10.1007/s10664-016-9490-0

[9] Wagner, S., & Deissenboeck, F. (2019). Defining Productivity in Software Engineering. Rethinking Productivity in Software Engineering, 29–38. https://doi.org/10.1007/978-1-4842-4221-6_4

INTERNATIONAL STANDARD SERIAL NUMBER INDIA

INNO SPACE SJIF Scientific Journal Impact Factor

doi cross ref

निस्केयर NISCAIR

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN SCIENCE | ENGINEERING | TECHNOLOGY

9940 572 462 6381 907 438 ijirset@gmail.com

www.ijirset.com

Scan to save the contact details