THE IMPACT OF AI-ASSISTED DEVELOPMENT ON THE PRODUCTIVITY OF SOFTWARE TEAMS

Teodora POPESCU[1]
Stefania GRECU[2]
Andreea-Georgiana GRIGOREANU[3]
Ana-Maria PERȘENEA[4]
Denisa-Florela MIRCEA[5]
Costin-Anton BOIANGIU[6]

Abstract

Now shaping routines in quiet ways, artificial intelligence enters routine coding tasks, altering how developers approach writing, reviewing, yet sustaining software across extended periods. Instead of fixating solely on technology, researchers examine links among AI-powered tools, personal views on output efficiency, confidence in automated decisions, while observing evolving patterns in team coordination inside tech teams. Data come from a field survey using standardized forms that collect scaled responses alongside open feedback, drawn from professionals varying in role, seniority, and frequency of AI tool usage. Though methods stay grounded in observation, emphasis rests on real voices rather than abstract trends. Each response adds texture to how automation weaves into daily practice, neither replacing nor leading but adjusting rhythms beneath the surface. Findings emerge not from theory alone, but from moments logged by those typing while assisted.

Guided by the SPACE model, this study looks at how people describe shifts in satisfaction, performance, task involvement, teamwork, and speed when using tools. At the same time, ideas from DORA help situate these experiences alongside known patterns in software delivery. Findings show those relying on AI coding aids often feel more productive and content, yet some note growing reliance on tool-generated advice. A quieter concern emerges thinking deeply about code may decline when responses come pre-filled. Risks around consistent results and reliance on output show up again and again. Wrapping up, attention shifts to how these tools fit carefully into what developers already do. One point

[1] Assistant Team Coordinator, POLITEHNICA National University for Science and Technology of Bucharest, Romania, teodora.popescu@stud.acs.pub.ro
[2] Student, POLITEHNICA National University for Science and Technology of Bucharest, Romania, stefania.grecu@stud.acs.pub.ro
[3] Student, POLITEHNICA National University for Science and Technology of Bucharest, Romania, andreea.grigoreanu@stud.acs.pub.ro
[4] Student, POLITEHNICA National University for Science and Technology of Bucharest, Romania, ana.persenea@stud.acs.pub.ro
[5] PhD Student, POLITEHNICA National University for Science and Technology of Bucharest, Romania, denisa.mircea@stud.acs.upb.ro
[6] PhD, Professor, POLITEHNICA National University for Science and Technology of Bucharest, Romania, costin.boiangiu@cs.pub.ro, corresponding author

stands out, human judgment needs structure, support, and defined roles where machines assist but do not decide.

Keywords: Artificial intelligence in software engineering, Perceived software productivity, AI-based coding assistants, Empirical developer survey, SPACE productivity framework, Team collaboration

JEL Classification : O31; O33; L86; J24; M15

1. Introduction

As artificial intelligence becomes part of everyday programming, fundamental aspects of building software begin to shift without announcement. Instead of standing apart, tools such as GitHub Copilot, ChatGPT, or Amazon CodeWhisperer blend into thought processes, suggesting snippets, untangling messy logic, simplifying complex concepts, catching overlooked issues. Because they are there, behavior adjusts how people write evolves, checking updates follows altered patterns, updating legacy code demands somewhat less effort. A quiet transformation settles in through repeated small changes. Because of this, groups start questioning older measures of progress. Speed of delivery or volume of output matters less than before. Attention turns toward mood during work, cognitive load across hours, patterns in collaboration between team members. Studies show artificial intelligence helpers shorten time spent on routine coding jobs with clear boundaries [1], but still, output in programming work rarely boils down to just one number. Not just about lines written or tasks done fast, the SPACE framework combines five parts: satisfaction, group collaboration, work delivery, performance, and efficiency, these are mixing outcomes with human elements [2]. Seen this way, it becomes clearer to separate clear-cut stats such as error rates or new features shipped from how developers personally experience advancement. Central here is not time logged, yet perceived productivity, the feeling programmers hold about moving ahead without struggle. Numbers may appear strong, still staff can describe hurdles, and people sense tends to split from recorded performance. Choosing personal understanding instead of numbers by themselves guides the research direction here.

Although some findings highlight benefits, others complicate the narrative around AI in software creation. Automating routine tasks, like basic testing or standard code segments, appears to speed up workflows, according to broad data reviews [3]. Yet real-world company observations reveal gaps in who gains. For junior coders, results often tilt positive; their output improves more clearly when aided by tools. Senior practitioners, however, experience subtler shifts, with outcomes differing across projects and teams. Added mental load and collaboration demands also surface under closer inspection [4], [5]. Not every coder sees their pace matching actual performance data, some feel faster without speed showing in numbers, occasionally slipping behind instead. From different workplaces, software engineers share thoughts via questionnaires, forming the base of this inquiry. Guided by concepts from the SPACE framework along with observations tied to DORA findings, attention turns to how builders of systems perceive artificial intelligence shaping what they produce, quality reached, and ways collaboration unfolds. Progress seems lopsided, the routine actions spark greater impressions of forward motion when machines assist, while deep or learning-intensive efforts lag in perceived boost. Time spent checking

machine-generated suggestions counts, but weighs less during straightforward chores, growing heavier as difficulty climbs. So, confidence swells clearly for easier work, yet blurs into uncertainty once challenges grow complex.

2. Conceptual Foundations and Literature Review

Something odd lately, coding helpers are not stuck to strict rules anymore, moving beyond old tricks like filling brackets or reusing snippets. Instead of just catching typos, they now guess what you are after, nudging decisions based on context. Pattern-driven models take charge, drafting bits of logic, walking through why a choice fits, shaping early ideas. These are not robots ticking boxes, so they act closer to someone who gets your goal. Tools like GitHub Copilot toss out starting points, trim time hunting docs. Oddly enough, speed does not always rise, sometimes thoughts slow down, tangled in double-checking, adjusting, second-guessing each hint that pops up. Actual improvements rely strongly on how hard the task is, how skilled the person is, along with the flow shared between individual and tool [6].

Evidence from sector analyses and wide-ranging professional feedback shows similar patterns. Where tasks repeat often and carry little uncertainty, like standard code blocks, docs, or minor rewrites, generative AI proves useful. However, performance drops when handling intricate, large-scale software challenges, mostly because unchecked AI contributions may harm consistency or correctness [7]. Lately, research emphasizes framing "smart coding aids" around their function within workflows rather than focusing on model types alone. Seen through utility, these tools fulfill four core purposes such as aiding execution (easing writing of code), easing mental load (offering insights or options), ensuring standards (finding flaws or validating tests), and enabling teamwork (sharing context via summaries or records).

Sometimes AI helpers act like coding partners, stepping into roles like a driver or guide during pair work, though people still handle key decisions and fit pieces together. Still, teamwork involving machines differs fundamentally from person-to-person cooperation. Without genuine awareness of context, artificial agents may generate responses that seem plausible yet miss the mark entirely. Their value emerges only when inputs are well framed, plus the user actively challenges what gets proposed. Viewed via distributed cognition theory, such technologies reshape how mental load, focus, and recall distribute across both coder and algorithm [8]. Even though many teams now use AI tools, confidence in them wobbles since too much faith invites blind trust and too little triggers endless checking. Data from recent surveys capture this split, so uptake grows fast yet doubts about precision and consistency linger [9].

Still common measures like code volume or frequency of updates fail to reflect real impact, so researchers now turn to SPACE, a model tracking personal insight, work standard, advancement speed, teamwork flow, and effort economy [10]. With artificial intelligence becoming part of daily coding routines, behavior patterns start revealing mental conditions including tiredness, pressure, or contentment; these inner signals frequently align with stronger results and better health. Yet outcomes across groups differ, showing clearest benefit only where smart systems fit naturally within current practices. Delivery benchmarks such as DORA add background detail about pace and reliability, even if they do not shape the core analysis.

Even so, mistakes pop up often when code is shaped by artificial intelligence. Output gains can vanish, especially if checking and using AI results takes too much time, depending on skill level, task complexity, or familiarity with current codebases [4], [6]. When supervision fails to keep pace with rollout, trusting suggestions without question may introduce errors or open doors to risks, particularly in weakly protected areas [11]. However, studies tracking long-term issues like defects, vulnerabilities, or lasting impact remain rare. Extended usage might quietly shift thought processes: attention drops, problem-solving lags, reliance grows, all unnoticed due to limited evidence. Over time, few studies follow key skills like judging system designs or spotting errors. Success factors tend to stretch past software tools, customs within teams and daily office patterns seldom get detailed attention. Although many reviews frame AI assistants as individual supports, outcomes depend heavily on collective norms, instruction styles, feedback cycles, along with joint troubleshooting efforts. Such omissions drive scholars to watch carefully how programming teams integrate artificial intelligence while working routine shifts.

3. The Landscape of AI-assisted Development

Change moves fast, while artificial intelligence advances quickly, introducing varied tools aiding programming at different stages. In past iterations, most features relied on static guidelines to assess code layout together with rudimentary completion aids. Current setups leverage expansive language frameworks, grasping nearby details to propose, explain, or adjust snippets with sharper awareness. Rather than operating strictly as mechanical helpers, they begin resembling collaborative thinkers, expanding individual coder potential. Clearly spotting this change allows a look at actual effects: productivity gains, altered collaboration patterns, cleaner, or occasionally more tangled, code. Results rely strongly on how tools are built, their integration depth within daily work, and user differences, given uneven adoption across firms and experience levels.

3.1. Overview of leading AI coding tools and platforms

Software creation today sees growing use of artificial intelligence tools meant to help write, understand, and manage code. While powered mostly by large language models, such systems vary widely, some reach deep into workflows, others sit lightly atop them; some span broad functions, while others stay narrow. Inside coding platforms like Visual Studio Code or JetBrains products, one familiar presence appears: GitHub Copilot [12], built first on Codex [13], then advanced through GPT-like architectures. It suggests lines or blocks during typing, using nearby text, notes, and surrounding files as cues. Tasks like filling repetitive structures, drafting tests, or shifting syntax between frameworks become quicker with its aid. Research findings [1] show gains in speed when work stays routine and well-defined. Yet alongside those benefits come concerns, generated logic may fail quietly, flaws might slip past notice, risky patterns could emerge, legal status often remains uncertain. Safeguards inside systems do not ensure accurate outcomes, so developers must still take charge of any code they bring in. As ChatGPT emerged [14], work methods began leaning into back-and-forth help shaped by prompts. Through web interfaces or embedded software, engineers turn to it for spotting mistakes, fixing logic flaws, creating small code pieces, weighing architectural choices, or shaping early drafts of guides. Since it cannot reach internal project storage, its strength lies in broad, flexible questions across topics, but there are no situations demanding detailed knowledge of a specific setup. Over time,

improvements like outside data links and longer recall slowly reduce those barriers, making the tool more adaptable through gradual change.

Unlike others, Amazon CodeWhisperer [15] focuses on enterprise needs, secure coding, adherence to policies, smooth connections across tools, offering instant recommendations, warnings about flaws, tracking of software licenses inside AWS environments and common development editors. On a parallel track, Google's Codey [16] appears alongside Microsoft Copilot within Azure DevOps [17], mirrored by proprietary large-scale models, each blending smart completion features with oversight mechanisms. Still, assistance driven by artificial intelligence shows uneven results, where outcomes hinge on how people engage, how deeply systems link together, and what tasks are targeted, gradually influencing teamwork patterns, collective processes, and levels of potential risk.

3.2. Adoption trends across organizations and skill levels

Though AI tools are spreading faster in software work, how fast depends on company size, field limits, and what skills teams hold [7], [18]. Bigger companies stick to approved licenses and clear rules. Smaller groups move slower, worried about who owns output, whether data stays private, and if results behave predictably. Skill level shapes use newer coders lean into AI when handling routine blocks, also tangled APIs, or syntax-heavy jobs because quick help lightens mental effort. Still, hesitation remains, fed by doubts around correctness, consistency, and how well code will age. When entire groups align on how they apply such tools, results generally improve. However, isolated usage creeps in, weakening uniformity across projects. Communication grows tense as styles diverge without common ground. More effort goes into syncing efforts when teamwork depends on mismatched habits.

3.3. Cognitive and behavioral changes in the development process

What counts most is not faster coding, but how AI reshapes a programmer's mindset. Shifts in thinking run deeper than time saved. New patterns of problem-solving emerge quietly. Efficiency fades beside changes in approach. The real change hides in daily habits. Tools alter judgment more than pace ever could. Because these systems offer instant coding help, attention patterns change, so do methods for tackling problems and interacting with existing code. Often, routine or syntax-heavy jobs feel easier mentally. Such relief fits ideas about managing mental effort when support matches ability, performance and understanding grow [19]. Studies show novice coders benefit most, spending less energy grasping or writing code. Yet relying on AI answers without scrutiny might weaken thorough thinking, slowing mastery over time.

Unexpectedly, too much trust in automated decisions can backfire. Studies of how people work with smart machines show that leaning heavily on them tends to dull attention, making mistakes harder to catch [20]. When coding, this habit might lead programmers to miss bugs in lines that look right or come with strong certainty. Instead of deep planning, some now shape their approach around tweaking everyday-language prompts step by step. While faster idea testing emerges from this method, real thought may shift outside the mind, weakening grasp of core logic when artificial support replaces personal analysis instead of strengthening it. When AI influences code, it becomes unclear whose work is whose, this confusion grows during group reviews, complicating accountability. Because tracing roots matters more now, documentation of choices throughout development phases becomes

essential for groups. While progress unfolds, keeping clear logs helps maintain clarity over time. Since oversight grows harder, attention to detail in recording steps supports better outcomes later. As work advances, consistent notes allow smoother reviews when needed most.

3.4. Organizational readiness and cultural adaptation

Even when teams prepare thoroughly, weak adaptation systems erode their readiness. Performance numbers matter less than whether tools align smoothly with daily work [21]. Smooth blending into workflows depends on unseen structure, roles defined, rules set, learning available. Where teams already follow steady DevOps routines, or operate under reliable supervision, AI slips in without resistance [22]. Absent such groundwork, attempts lose momentum fast. Gains vanish just as quickly. What really shapes uptake isn't policy, but shared belief if trying new things carries little risk, minor missteps become lessons instead of setbacks. Progress holds when people know it's okay to stumble now and then. Yet pressure to perform perfectly pushes practice underground, into isolated fixes that never connect. Learning grows best through steady encouragement, which quietly balances excitement with thoughtfulness, a subtle guard against going too far [24]. Facing intense oversight, companies now reshape rules about content ownership and personal information handling, clarity turns essential [25]. Shifts that endure come less from devices, more from how people work together differently, so these adjustments uphold system reliability while aligning group actions.

4. Research Design and Methodology

4.1. Overall research strategy and rationale

Beginning with how groups apply artificial intelligence in coding tasks, the work draws from combined insights, scores given by users alongside personal accounts they shared. Instead of relying on lab settings or real-time monitoring, patterns emerge through statistics merged with free-text comments provided by participants. Human interaction shapes software creation because of that, methods mixing numbers and narratives fit better into diverse workplace routines. Recording every keystroke or decision made within a group seldom happens beyond academic proposals [26].

One way to see change is through the eyes of developers using AI tools every day compared to those stepping back or just starting. One path leads through pace, another through how people work together, both shape what clean code means in practice. Lived moments build patterns slowly, not loud but steady, shaped by who uses tools often, where they work, and their growing abilities. Productivity once ruled research, but now a quieter thought holds ground and measuring change needs many angles. What people do on the job grows out of overlapping actions, repeated patterns, mental demands, also quiet cues from surroundings. Since such elements combine in unpredictable ways, specialists view frameworks such as SPACE and DORA less as rigid scores, more as flexible views that show distinct features once put into practice [22].

4.2. Data collection procedures and tools

Among those surveyed, software professionals came from varied job levels and company types. Chosen because it supports broad reach without identifying participants, the digital questionnaire helped gather responses efficiently. Insights into how developers view

artificial intelligence in their work emerged clearly through this method, especially across differing workplace settings [27].

Not everyone joined, only those willing stepped forward. Teams arrived from varied corners of industry as fledgling tech ventures, established companies, growing businesses, and freelance advisors. Engineers who build systems, testers who check quality, operations experts managing deployments, all shared their views. Some had just started careers. Others brought ten or more years behind screens. Workplaces ranged from shops using artificial intelligence daily to places sticking with classic methods. Names stayed hidden and nothing tracked back to individuals. Each person learned upfront what the research aimed to explore and how long answers might take. Volunteers chose freely, knowing purpose and scope before contributing anything.

Broken into parts that match the research model, the survey covers several areas. First come details about age, job role, and work history, followed by how people use artificial intelligence tools across different stages of development. Instead of listing every option, questions explore personal views on speed and output improvements at work. Shifts in coding standards and career progress show up next, through individual impressions rather than metrics. Then comes group interaction: how colleagues react, whether they rely on machine suggestions, and how sure they feel using them. Open answers dig into concerns and changes in daily habits over time. Some items ask for fixed choices to build statistical clarity, others invite reflection for richer context. What emerges is a mix of data and stories shaped by how people took part.

4.3. Data analysis techniques and validation methods

A closer look at the data began with structured responses, their figures forming a base. Yet beyond these totals, comments added context, revealing motives tucked inside words. Because hard metrics alone could miss nuance, reflections helped fill gaps. When trends appeared in charts, stories explained why they mattered. Numbers gained meaning after pairing with moments people described. Only then did connections become visible, between what was measured and lived.

Figures stepped in for categories and review scores so standard math tools could work. How folks perceive situations shapes these results, rather than real-world facts. Focus shifted toward AI usage levels, job experience length, and where people worked when comparing clusters. A central idea stands, personal impressions guide findings, not fixed data points. At times, data followed expected patterns, making Welch's t-test suitable; when distributions tilted or involved ranks, Mann–Whitney U stepped in. For examining connections between coding frequency and work quality, Chi-square detected associations, then Cramer's V measured their strength. Trust and satisfaction levels, measured on an ordinal scale, led to choosing Spearman along with polychoric correlation methods for uncovering hidden connections [28], [29]. When sample sizes or statistical conditions didn't align, Kruskal–Wallis took the place of ANOVA to explore skill-related variations. From qualitative responses emerged repeated ideas, such as mental effort, dependence on tools, and difficulties in teamwork, noticed during initial review. These observations later connected with numerical trends through combined analytical approaches. Results gained stronger support by aligning them with SPACE and DORA models, showing steady changes in performance tied to real-world actions while reducing interpretive distortions [30].

4.4. Data sources and metrics

Despite relying on self-reported responses, this study ties its assessment of AI-supported coding practices to recognized models for output and deployment success. Because software work involves both people and systems, the chosen measures reflect not only how developers rate their own efficiency but also how they view teamwork and personal workflow impacts. Productivity here emerges through individual judgment, shaped by real project conditions and interaction patterns within teams.

Measuring productivity relied on the SPACE framework. Using this approach, survey questions got sorted into five areas. One area looked at how people felt about their work-life balance and job satisfaction. Another focused on personal ratings of performance when using AI support. How often tools appeared across different stages of building software formed a third category. Team dynamics, how coworkers communicated and worked together, made up the fourth part. Time saved each day due to automation rounded out the set.

How fast changes feel. Instead of hard data, developers shared views on how quickly features go live, this stands in for lead time. Deployment rhythm came through their sense of release pace, not system records. When systems broke, recovery duration was judged by personal experience, often shaped by AI tools helping fix issues. Code stability perceptions formed the basis for failure rates after AI involvement. No backend logs fed into these measures, every insight stems from individual viewpoints. What shows up here reflects opinion, not operational traces.

Although numbers fail to show everything, feelings of simplicity, consistency, and natural alignment emerged uniquely within each group [31]. From how people exchanged messages while building and reviewing work, hidden ways of acting came into view. Stories shared by individuals brought up attachment to owning ideas, uncertainty in assessing teammates' efforts with AI support, reliance on what machines produce without question. Because of this blend, weighing quick results against possible downsides, like higher upkeep demands, unclear responsibility lines, weaker bonds among members, became feasible. The resulting view depends less on software used, more on how people react within those settings.

5. Interpretation of Key Performance Indicators

Looking at how software teams perform, this section weaves together data drawn directly from practitioners working in the field. While examining AI's part in coding tasks, attention shifts toward how speed, teamwork, and confidence in decisions shift when tools assist developers. Instead of treating numbers alone as truth, findings pair hard statistics with personal accounts collected through interviews. Because context shapes what success means, each measure is weighed alongside real-world conditions faced by engineers daily. One insight stands clear, productivity cannot be boxed into a single score without losing meaning. Outcomes shift quietly, routine boundaries soften once workflow design meets leadership decisions, yet people stay in control. Adjustment shows up instead of full substitution, shaped by hands-on insight that numbers alone cannot capture [33].

5.1. Quantitative results: task efficiency, output quality, and review dynamics

A clearer sense of how tasks perform emerges when personal assessments of output quality mix with estimates of time cut by artificial intelligence, particularly where broad oversight fails [34]. Where regular and occasional users differ, the gap might not reflect real division, chance alone could shape it. Sorting signal from noise led researchers toward techniques that judge whether gaps likely persist past random chance. Conclusions only strengthen after uncertainty fades into background. Efficiency takes shape not in absolutes, but in measured doubt.

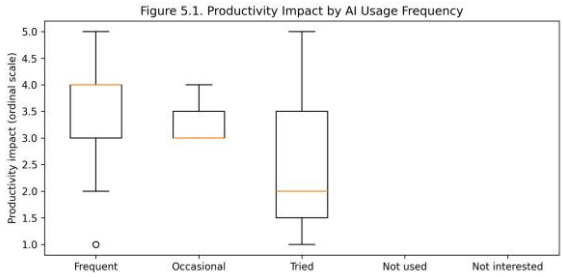


Figure 5.1 (Productivity impact by AI usage frequency) illustrates that people using AI regularly give higher median productivity ratings. On the opposite end, individuals trying the technology just once show results spread far apart, less consistency overall. Those in between, with some but limited use, fall modestly above the rest. Differences appear noticeable, though not strong enough to treat as definitive here. A Welch t-test was run on the ranked responses from 1 to 5. This analysis compared two specific cohorts as frequent AI users (n = 19) against a non-frequent group (including both occasional and trial users, n = 10). While the frequent-use group did report a higher mean productivity score (M = 3.53, SD = 1.02) than the non-frequent group (M = 3.10, SD = 1.10), the difference failed to reach statistical significance: t (17.22) = 1.02, p = 0.323. The size of the effect turned out small (d = 0.41), pointing toward only a slight link when usage happens often and better results. Because rankings were involved, researchers used a Mann–Whitney U test afterward, its outcome stayed in line with earlier data (U = 120.5, P = 0.226) yet failed to settle uncertainty.

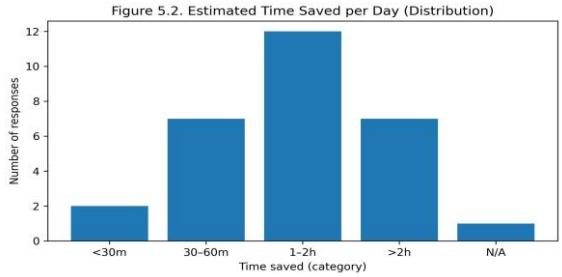


Figure 5.2 (Estimated time saved per day) indicates that most people using AI tools say they gain more than two hours each day, yet some find only thirty minutes to two hours back in their schedule. Despite differences in how much time is saved, nearly everyone notices a boost in what they get done every day, since hardly anyone says there is no real advantage, this points to changes deeper than just small tweaks. When it comes to work results,

confidence grows not only in correct code but also in how well it can be updated later and whether it holds up over time.

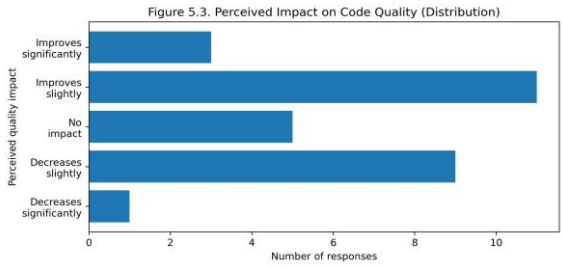


Figure 5.3 (Code quality impact distribution) indicates slight improvements in code quality for several users, a few observe worsening outcomes. Underlying problems might include overlooked bugs, excessive dependence on automation, or heavier mental demands when examining code. Such factors reveal that advantages are sometimes balanced by hidden complications.

AI Usage Level / Code Quality	**Decreases (significantly decreases)**	**Improves (significantly improves)**	**Neutral (no impact)**	**Row Total**
Frequent use	6	10	3	19
Occasional use	2	3	2	7
Tested only	2	1	0	3
Column total	10	14	5	29

Table 5.1. AI Usage Level and Perceived Code Quality (Aggregated Categories, N = 29)

Even with limited participants and scattered results, AI reliability factors split into two groups: those using it often versus those using it rarely. Shown in Table 5.1, a 2×3 layout connects frequency of use to three result types as better, stable, or poorer, making patterns easier to grasp. Because of this structure, shifts in performance become more transparent. Trends stay intact, free from skewing influences, when examined through this frame.

AI Usage Levels / Code Quality	**Decreases**	**Improves**	**Neutral**	**Row Total**
Frequent use	6	10	3	19
Non-frequent	4	4	2	10
Column total	10	14	5	29

Table 5.2. Frequent use and Non-frequent and Perceived Code Quality

AI Usage Levels / Code Quality	Decreases	Improves	Neutral
Frequent use	6.5517	9.172	3.276
Non-frequent	3.448	4.828	1.724

Table 5.3. Performance fluctuations: A comparative analysis of frequent vs. infrequent AI usage

When exploring if perspectives on code quality change alongside AI usage habits, responses fell into three categories: better, unchanged, or worse. Because of small sample size (N = 29), several cells in the initial 3×3 layout held very few cases, especially those labeled "tested only." To fix instability, researchers combined occasional users and testers into one low-use group, forming a clearer 2×3 structure. Still, statistical analysis revealed little link between regular AI use and perceived code quality; chi-square results gave $\chi^2 = 0.418$ and $p = 0.811$. Supporting that finding, Cramer's V stood at 0.12, a minimal strength of association, suggesting although individuals reported varied experiences, trends across usage types do not hold consistently within this set of respondents.

5.2. Qualitative insights: perceptions, trust, and human factors

Upon reviewing past statements, a pattern emerged where programmers rely on artificial intelligence to ease thinking and accelerate routine actions yet trust in its suggestions differs sharply among users. Some express strong faith: others remain doubtful. Speed is frequently mentioned in comments as a goal, but beneath lies tension in human-machine collaboration: depending too much on automation risks poor decisions just as much as ignoring it outright disrupts clarity [34]. That tightrope between dependence and doubt emerged as a key thread through interview remarks. Belief must be earned. Choices remain debated.

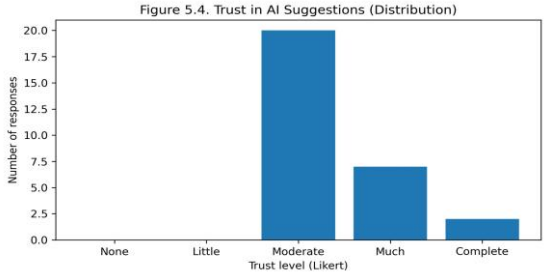
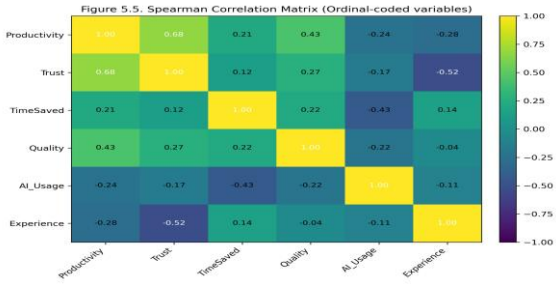


Figure 5.4 (Trust distribution) most answers fall into the middle range of trust, suggesting people adjust their faith in AI rather than fully embrace or reject it. Extreme positions, whether blind acceptance or complete doubt, appear infrequently across responses. Now seen everywhere, grasping how people perceive machine responses matters more, viewing them not as facts, yet uncertain guidance. Evidence suggests these answers often face review prior to application, shaped by individual knowledge. Depending on situation, trust changes and basic programming jobs tend draw higher reliance. Still, doubt remains regarding correctness, sparking unease around careless reasoning if suggestions are accepted too easily. This discomfort causes hesitation, though results appear solid. User comments point to constant tweaks, working systems into daily flow while keeping

authority. Movement follows movement, never pausing. Judgment waits at the final stage, unmoving.



relationships among central ideas emerge through Likert-based assessments; rank-order consistency matters more than equal spacing, making Spearman ideal despite non-linear spacing. With few participants involved, interest shifted from statistical significance toward patterns showing clear direction and real-world meaning. Throughout the visual map, a single pattern stands out, that is the greater reported trust links to improved judgments of quality, reduced effort, along with heightened perception of speed, each increasing together. Older engineers usually question AI advice more than others do. How people see dependability, their sense of possible downsides, and mental effort matter just as much as correctness. Lived practice counts heavier than claims about faster work or more results.

5.3. Comparative patterns by experience level and project type

Right away, how people experienced AI depended heavily on their skill level and the situation they were in, developers saw different outcomes [33]. For those new to coding, assistance from AI often leads to quicker progress, reducing uncertainty during early learning phases. Supportive feedback loops form when suggestions guide decisions, helping novices build confidence without getting stuck too long. Meanwhile, experienced programmers report less consistent gains, some even find the process slows them down. Gaps open when automated output demands review, interrupting rhythm developed through years of routine. Familiar patterns clash with unfamiliar prompts, creating friction rather than flow. Some groups advance more slowly than others. Yet results frequently fall short of what was hoped.

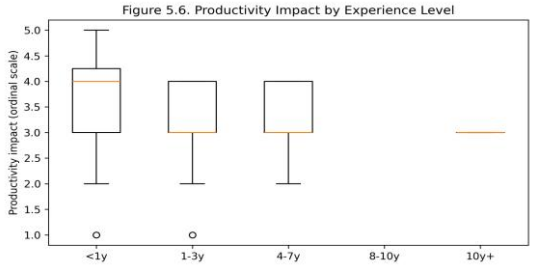


Figure 5.6 (Productivity impact by experience level) illustrates this moderating pattern, a trend I observed when synthesizing the disparate gains across career stages, where junior developers report the most substantial median productivity improvements, followed by a more modest uptick among mid-level staff. With only a few participants among the most experienced group, conclusions about veteran developers remain uncertain, yet signs point

to a leveling off. Though limited, the pattern suggests gains taper off later in their careers. Shifting demands, like tighter quality controls, drove up time spent reviewing work. That extra scrutiny came at a cost and faster progress gave way to stronger design foundations. Performance varies too much to assume consistent results across settings. The environment shapes outcomes more than experience alone.

In this analysis, the ANOVA compared three experience groups (<1 year, 1–3 years, 4–7 years), excluding the >10 years category because n = 1. Thus, $k = 3$ and $N = 28$, yielding:

$df_{(\text{between})} = 2$ and $df_{(\text{within})} = 25$, with $F(2, 25)$. Let $\bar{x} = \Sigma x_{ij} / N = 3.393$ and $SSbetween = 1.576$ and $SSwithin = 29.1$. Hence: $MSbetween = SSbetween/ dfbetween = 0.788$ and $MSwithin = SSwithin/ dfwithin = 1.164$ so that: $F = MSbetween/ MSwithin = 0.68$, $\eta^2 = 0.051$, H=1.82 and p=0.402.

Addressing the possibility of variance across career stages, I obtained $F(2, 25) = 0.68$ with a corresponding $p = 0.517$, results that preclude any claim of statistically significant differences. These figures, notably characterized by a small effect size of $\eta^2 = 0.051$, suggest that the experience groups do not diverge in a measurable way. Instead of relying solely on parametric methods, a different route emerged: applying the Kruskal–Wallis test suited for ranked outcomes. That alternative approach also found no notable separation among categories ($H = 1.82$, $p = 0.402$). Differences spread too thin when viewed as a whole, sharp calls or firm takeaways can't follow from such scattered results.

5.4. Cross-validation of quantitative and qualitative evidence

Though numbers offered some insight, they failed to show the full picture, so researchers combined methods to uncover mismatches between data and reality [32]. Validity of improvements depended on actual assistance available in relevant cases, relationships between daily routines and reported results were probed using Chi-square tests [28]. Context matters deeply in managing teams with AI, because how people think and interact often steers outcomes more than metrics suggest [33].

6. Framework for Responsible AI Integration in Software Teams

6.1. Principles for ethical and sustainable use of AI assistants

What first caught attention wasn't the system's performance, yet tiny mismatches in early outputs revealed underlying flaws. Over months, influence grew not from algorithms alone, but from shifts in team routines [20], [25], [33], altering tool adaptation gradually. From this emerged a method shaped around five ideas. At its heart sat openness and supervision, applied via clear signs when AI contributed (such as notes in commits or tags in requests), ensuring actions stayed trackable [25], and still upholding that people must answer for outcomes. A method took shape, "humans always involved unless otherwise removed", where outputs from artificial intelligence count only as preliminary sketches. From my standpoint, compromise here is not an option; code demands inspection, validation, and personal liability, especially once vulnerabilities appear or reasoning grows intricate [21], [33].

Although less visible, fairness mattered deeply, so choices could not tilt toward more experienced roles. Instead of counting hours alone, performance drew from multiple angles, how fast work finished, whether it stayed accurate, if others contributed smoothly, and even

how people felt during the process. Hidden risks needed attention too, which is why digital safety steps came next, privacy rules tightened control over information flow. Automated checks hunted for exposed secrets, while isolated systems limited exposure. Monitoring did not stop at launch and ongoing tracking ensured issues surfaced before they grew. Shifting impacts of AI mean relying on both DORA metrics and human elements from SPACE guides how I judge things. Without frequent check-ins, understanding stays shallow, figures alone miss too much. Though data helps, it cannot carry the full weight. What matters emerges only through ongoing reflection. Moments of review reveal what static reports hide. Context shapes how we see exact numbers. As choices unfold in real time, weaknesses in oversight start showing up.

6.2. Balancing automation, oversight, and human creativity

Most of the time, automation makes sense after basics are clear: when it comes to repeating actions like filling in code frames, small corrections, or standard checks, artificial intelligence works well, yet planning structure or solving intricate logic still depends on people [20, 33]. For vital parts, oversight by a person remains necessary since models often lack awareness of background details, particularly during uncertain or shifting conditions.

Few realize how quietly inspection shapes today's standards. Oversight slips in, not by choice but by need. Rather than accepting artificial intelligence outputs at face value, the process treats them merely as initial drafts. Each result faces review like human-generated work, through comparison with peers, repeated testing, sometimes close inspection of reasoning paths. Prompted by unease around unquestioning faith in fluent responses, this layer challenges assumptions that slick presentation means accuracy [20]. Used as a device to stretch thought during concept shaping, not to override decisions, I aimed to keep core human abilities intact. In the end, even when automation manages scale, control over creation stays with the engineer.

6.3. Organizational implications for Agile and DevOps practices

Because alignment matters, this framework ties AI use straight into current Agile and DevOps practices. From working alongside Agile groups, speeding up parts of development rarely cuts down time needed later for checks or merging work. So, viewing AI-written code as extra bandwidth tends to overload teams while distorting how fast progress really is. Instead, I built space for thorough verification right into sprint forecasts, making sure effort predictions cover everything from start to stable deployment, not only the first draft.

Because AI alters how teams work, old measures like delivery speed and error rates face new pressures. When moving faster, stability can slip unless safeguards stay strong. Every update shaped by artificial intelligence now moves through regular testing lanes before going live. Checks for performance and errors remain active at each stage. This keeps quality steady even when tools accelerate output [22].

Junior developers now turn to AI for support, whereas seniors spend greater time shaping system structure and overarching plans. Review methods shifted accordingly, less attention goes to detailed code inspection, more to architecture and potential risks. A mix of Agile, DORA, and SPACE measures brings clarity, revealing effort once buried in teamwork and validation tasks. What was overlooked before now shows up clearly.

6.4. Training and continuous learning strategies

Now it takes more than luck to make real progress, planning matters. Skills must grow alongside tools, shaped by design rather than chance. Writing prompts well becomes essential, since coding changes from start to finish when machines join the process. Reviewing shifts, thinking adapts, learning deepens, all affected differently. Falling short in skill while tools advance is a gap that widens fast. Progress demands matching pace, each leap met with equal learning.

Young programmers today grow both foundational abilities and keener thinking at once. Code written by automated tools offers a way to examine results alongside built-in logic, walking through functions one at a time, this often uncovers problems tucked beneath the surface. Not meant to replace insight, such technology strengthens comprehension when treated as a mirror for individual thought. Skilled programmers tend to follow another route entirely and combining human judgment with machine output matters most during tasks where smart software sharpens known abilities. Evaluating design choices or simplifying how files are structured appear here, tackling concerns such as maintenance over years and closer inspection directly rather than looking away.

Only later do we see it, growth hides in daily work, such as checking code or discussing plans once a sprint finishes. Errors surface, then become learning moments because someone looks closer. Thoughts move from one person to another like folded messages slipped beneath a doorway: unspoken yet clear. What people expect changes bit by bit, formed more by routine experience than big declarations. Over time, clarity emerges, grasp of artificial intelligence builds slowly, never in one fixed session. While operating, its shape adapts. New fragments enter via short bursts of data. Almost at once, changing documents adjust in parallel. Motion happens during use.

6.5. Risks, limitations, and contextual factors

When used beyond labs, artificial intelligence operates within limits, not because of the technology's flaws, but due to tangled workflows inside institutions. Shifts in results emerge slowly, shaped more by how teams function than by code upgrades. Hidden routines influence decisions, often overriding system capabilities. Readiness gaps dull impact, even when tools perform well on paper.

A first concern showed up fast, automation bias. Sometimes developers trust outputs simply because they look polished, though flawed underneath, inviting unseen errors or weak spots in safety [20]. Still, issues grow with broad-use models, when accuracy matters, these systems falter, lacking memory of earlier decisions or how teams operate, so support shifts toward confusion [4], [9]. Starting fast may end slow, effort piles up, outcomes lag working solo. This approach presumes zero at every stage. A single result steps into review, treated as uncertain without verification. Following alignment with recognized standards like ISO/IEC [25], progression becomes possible.

Culture influences results as powerfully as tech, though that link slips by unnoticed. Where duties are shared, movement happens, no pushing needed. Conversations nudge change, soft yet steady. A tool’s success ties closely to how people interact each day. Slow erosion of trust opens space for unseen problems to emerge. Since artificial intelligence helpers operate across wider systems, success depends on smooth fit with everyday routines instead of standalone performance. Improvement tends to come less from raw capability more from matching real-world settings. Over time, awareness grows slowly, continuous watching

matters when seeing how well these tools adjust. While tracking change, attention reveals patterns that only appear with patience.

7. Conclusions

Though speed improved, teamwork and code standards showed little improvement. Efficiency seemed higher at first, yet slowed down once reviews began. The mental effort moved elsewhere instead of disappearing, sometimes growing when checking and polishing work. Results varied widely depending on team and situation. Gains were not reliable across different settings. A closer look reveals that early gains in time frequently reappeared later once manual oversight became necessary. Shifting efficiency in one area could introduce hurdles elsewhere, particularly where responsibilities crossed paths. Identical tools were used, yet outcomes varied sharply based on experience. Automation tends to support beginners working through code, whereas seasoned professionals managing intricate setups see little gain. Such patterns highlight a core idea, effectiveness depends heavily on environment, position, and system complexity. What matters is not the tool itself but where it fits within daily work.

Still, early signs back the main idea, but just barely, when seen through careful, situation-specific scrutiny. Though interviews and polls agree tasks repeat faster thanks to tools, numbers tell a scattered story, rarely reaching firm conclusions across measures. What stands out instead is not broader efficiency, yet a shift in how mental energy gets spent. As jobs grow harder, checking work slows everything down. Oddly enough, people using these systems often report feeling quicker, even if clocks say otherwise, or worse, reveal delays hiding beneath smooth interactions.

Theoretically, this work underlines why AI-enhanced engineering should be seen through a combined social and technical lens. Merging the people-focused SPACE model with performance indicators drawn from DORA leads to one conclusion: measuring AI's effect demands context, its influence unfolds where individuals, systems, and structures meet. In real-world terms, findings suggest caution, an alert rather than applause for automation. Code produced by artificial intelligence ought to assist, not replace, human judgment. Oversight stays central, along with updated integration pipelines and learning paths built for varying skill levels. Absent these measures, gains in speed might only hide deeper flaws, a slow decline in software integrity, paired with weakened collaboration across teams.

Toward the edge of change, specialized AI agents seem bound to weave into daily work routines. Still, the real test for today's institutions lies less in processing power and more in how carefully we manage reliance on such tools. Keeping control in human hands matters, so does holding fast to sound engineering methods amid rising automated support. Studies down the line ought to follow actual outputs over time, watching how teamwork and skill-building shift across months or years. Hope stays guarded: when framed as a teammate instead of a replacement, artificial intelligence might lift output meaningfully while leaving room for people to lead.

References

[1] S. Peng, E. Kalliamvakou, P. Cihon, and M. Demirer, "The Impact of AI on Developer Productivity: Evidence from GitHub Copilot", Working Paper, Feb. 2023.

[2] B. Houck, T. Lowdermilk, C. Beyer, and S. Clarke, "The SPACE of AI: Real-World Lessons on AI's Impact on Developers", Industry Report, Aug. 2025.

[3] McKinsey & Company, "Unleashing Developer Productivity with Generative AI", McKinsey & Company Report, Jun. 2023.

[4] Z. Cui, M. Demirer, S. Jaffe, L. Musolff, S. Peng, and T. Salz, "The Effects of Generative AI on High-Skilled Work: Evidence from Three Field Experiments with Software Developers", Working Paper, Feb. 2025.

[5] J. Becker, N. Rush, B. Barnes, and D. Rein, "Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity", Industry Report, Jul. 2025.

[6] P. Vaithilingam, T. Zhang, and E. L. Glassman, "Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models", in Proc. CHI Conf. Human Factors in Computing Systems Extended Abstracts, Apr. 2022.

[7] McKinsey Global Institute, "The Economic Potential of Generative AI: The Next Productivity Frontier", McKinsey Global Institute Report, Jun. 2023.

[8] E. Hutchins, _Cognition in the Wild_, MIT Press, Cambridge, MA, 1995.

[9] Stack Overflow, "Developer Survey Results 2023", Stack Overflow Inc., 2023.

[10] N. Forsgren, M.-A. Storey, C. Maddila, T. Zimmermann, B. Houck, and J. Butler, "The SPACE of Developer Productivity", Communications of the ACM, vol. 64, no. 11, pp. 62–69, Jun. 2021.

[11] H. Pearce, B. Ahmad, B. Tan, B. Dolan-Gavitt, and R. Karri, "Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions", arXiv preprint, Dec. 2021.

[12] GitHub, "GitHub Copilot: Your AI Pair Programmer", GitHub Documentation, 2024. [Online]. Available: https://github.com/features/copilot

[13] OpenAI, "Codex: An AI System for Translating Natural Language to Code", OpenAI Blog, Aug. 2021. [Online]. Available: https://openai.com/blog/openai-codex

[14] OpenAI, "API Platform Overview", OpenAI Documentation, 2024. [Online]. Available: https://platform.openai.com/docs/overview

[15] Amazon Web Services, "Amazon CodeWhisperer: AI Coding Companion", AWS Documentation, 2024. [Online]. Available: https://aws.amazon.com/codewhisperer/

[16] Google Cloud, "Codey: Generative AI for Developers", Google Cloud Documentation, 2024. [Online]. Available: https://cloud.google.com/vertex-ai/docs/generative-ai/code

[17] Microsoft, "GitHub Copilot for Business and Enterprise", Microsoft Learn, 2024. [Online]. Available: https://learn.microsoft.com/github/copilot/copilot-for-business

[18] Stack Overflow, "Developer Survey 2024: AI Tools and Developer Productivity", Industry Report, Stack Overflow, 2024. [Online]. Available: https://survey.stackoverflow.co/2024/ai

[19] J. Sweller, "Cognitive Load Theory", Psychology of Learning and Motivation, vol. 55, pp. 37–76, 2011.

[20] R. Parasuraman and V. Riley, "Humans and Automation: Use, Misuse, Disuse, Abuse", Human Factors, vol. 39, no. 2, pp. 230–253, Jun. 1997.

[21] E. L. Trist and K. W. Bamforth, "Some Social and Psychological Consequences of the Longwall Method of Coal-Getting", Human Relations, vol. 4, no. 1, pp. 3–38, 1951.

[22] N. Forsgren, J. Humble, and G. Kim, _Accelerate: The Science of Lean Software and DevOps_, IT Revolution Press, 2018.

[23] F. D. Davis, "Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology", MIS Quarterly, vol. 13, no. 3, pp. 319–340, Sep. 1989.

[24] T. H. Davenport and R. Ronanki, "Artificial Intelligence for the Real World", Harvard Business Review, vol. 96, no. 1, pp. 108–116, Jan.–Feb. 2018.

[25] ISO/IEC, "ISO/IEC 23894: Information Technology – Artificial Intelligence – Risk Management", International Standard, 2023.

[26] M. A. Storey et al., "Guiding Principles for Using Mixed Methods Research in Software Engineering", IEEE Software, 2025.

[27] B. Kitchenham and S. Charters, "Guidelines for Performing Systematic Literature Reviews in Software Engineering", EBSE Technical Report EBSE-2007-01, 2007.

[28] A. Field, _Discovering Statistics Using IBM SPSS Statistics_, 4th ed., Sage, 2013, pp. 510–658.

[29] F. P. Holgado-Tello, S. Chacón-Moscoso, I. Barbero-García, and E. Vila-Abad, "Polychoric Versus Pearson Correlations in Exploratory and Confirmatory Factor Analysis of Ordinal Variables", Quality & Quantity, vol. 44, no. 1, pp. 153–166, 2010.

[30] T. D. LaToza and A. van der Hoek, "Crowdsourcing in Software Engineering: Models, Motivations, and Challenges", IEEE Software, vol. 33, no. 1, pp. 74–80, 2016.

[31] S. Amershi et al., "Guidelines for Human-AI Interaction", in Proc. CHI Conf. Human Factors in Computing Systems, 2019, pp. 1–13.

[32] J. W. Creswell, _Research Design: Qualitative, Quantitative, and Mixed Methods Approaches_, 4th ed., SAGE Publications, 2014.

[33] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, _Experimentation in Software Engineering_, 2nd ed., Springer Nature, 2024.

[34] J. D. Lee and K. A. See, "Trust in Automation: Designing for Appropriate Reliance", Human Factors, vol. 46, no. 1, pp. 50–80, 2004.

Bibliography

AMERSHI S., WELD D., VORVOREANU M., FOURNEY A., NUSHI B., COLLISSON P., SUH J., BENNETT P. N., INKPEN K., TEEVAN J., KIKOSKI B., HORVITZ E. – Guidelines for Human-AI Interaction – In Proc. CHI Conference on Human Factors in Computing Systems. [pp. 1–13]. 2019.

BECKER J., RUSH N., BARNES B., REIN D. – Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity – Industry Report. [no pages]. July 2025.

CRESWELL J. W. – Research Design: Qualitative, Quantitative, and Mixed Methods Approaches – ISBN 978-1-4522-2609-5. SAGE Publications. [no pages]. 2014.

CUI Z., DEMIRER M., JAFFE S., MUSOLFF L., PENG S., SALZ T. – The Effects of Generative AI on High-Skilled Work: Evidence from Three Field Experiments with Software Developers – Working Paper. [no pages]. February 2025.

DAVENPORT T. H., RONANKI R. – Artificial Intelligence for the Real World – Harvard Business Review, vol. 96 no. 1. ISSN 0017-8012. [pp. 108–116]. Jan.–Feb. 2018.

DAVIS F. D. – Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology – MIS Quarterly, vol. 13 no. 3. ISSN 0276-7783. [pp. 319–340]. September 1989.

FIELD A. – Discovering Statistics Using IBM SPSS Statistics – ISBN 978-1-4462-4918-5. Sage Publications. [pp. 510–658]. 2013.

FORSGREN N., HUMBLE J., KIM G. – Accelerate: The Science of Lean Software and DevOps – ISBN 978-1-942788-33-1. IT Revolution Press. [no pages]. 2018.

FORSGREN N., STOREY M.-A., MADDILA C., ZIMMERMANN T., HOUCK B., BUTLER J. – The SPACE of Developer Productivity – Communications of the ACM, vol. 64 no. 11. ISSN 0001-0782. [pp. 62–69]. June 2021.

HOLGADO-TELLO F. P., CHACÓN-MOSCOSO S., BARBERO-GARCÍA I., VILA-ABAD E. – Polychoric Versus Pearson Correlations in Exploratory and Confirmatory Factor Analysis of Ordinal Variables – Quality & Quantity, vol. 44 no. 1. ISSN 0033-5177. [pp. 153–166]. 2010.

HUTCHINS E. – Cognition in the Wild – ISBN 978-0-262-58146-2. MIT Press, Cambridge (MA). [no pages]. 1995.

KITCHENHAM B., CHARTERS S. – Guidelines for Performing Systematic Literature Reviews in Software Engineering – EBSE Technical Report EBSE-2007-01. [no pages]. 2007.

LATOZA T. D., VAN DER HOEK A. – Crowdsourcing in Software Engineering: Models, Motivations, and Challenges – IEEE Software, vol. 33 no. 1. ISSN 0740-7459. [pp. 74–80]. 2016.

LEE J. D., SEE K. A. – Trust in Automation: Designing for Appropriate Reliance – Human Factors, vol. 46 no. 1. ISSN 0018-7208. [pp. 50–80]. 2004.

MCKINSEY & COMPANY – Unleashing Developer Productivity with Generative AI – McKinsey & Company Report. [no pages]. June 2023.

MCKINSEY GLOBAL INSTITUTE – The Economic Potential of Generative AI: The Next Productivity Frontier – McKinsey Global Institute Report. [no pages]. June 2023.

PARASURAMAN R., RILEY V. – Humans and Automation: Use, Misuse, Disuse, Abuse – Human Factors, vol. 39 no. 2. ISSN 0018-7208. [pp. 230–253]. June 1997.

PEARCE H., AHMAD B., TAN B., DOLAN-GAVITT B., KARRI R. – Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions – arXiv preprint. [no pages]. December 2021.

PENG S., KALLIAMVAKOU E., CIHON P., DEMIRER M. – The Impact of AI on Developer Productivity: Evidence from GitHub Copilot – Working Paper. [no pages]. February 2023.

STACK OVERFLOW – Developer Survey 2024: AI Tools and Developer Productivity – Industry Report, Stack Overflow. [no pages]. 2024.

STACK OVERFLOW – Developer Survey Results 2023 – Stack Overflow Inc. [no pages]. 2023.

STOREY M. A., FORTIN M., ARANDA J., ZIMMERMANN T., RICHARDSON I. – Guiding Principles for Using Mixed Methods Research in Software Engineering – IEEE Software. ISSN 0740-7459. [no pages]. 2025.

SWELLER J. – Cognitive Load Theory – Psychology of Learning and Motivation, vol. 55. ISSN 0079-7421. [pp. 37–76]. 2011.

TRIST E. L., BAMFORTH K. W. – Some Social and Psychological Consequences of the Longwall Method of Coal-Getting – Human Relations, vol. 4 no. 1. ISSN 0018-7267. [pp. 3–38]. 1951.

VAITHILINGAM P., ZHANG T., GLASSMAN E. L. – Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models – In Proc. CHI Conference on Human Factors in Computing Systems Extended Abstracts. [no pages]. April 2022.

WOHLIN C., RUNESON P., HÖST M., OHLSSON M. C., REGNELL B., WESSLÉN A. – Experimentation in Software Engineering – 2nd ed., Springer Nature. ISBN 978-3-319-91437-7. [no pages]. 2024.

Amazon Web Services – Amazon CodeWhisperer: AI Coding Companion – AWS Documentation. [online resource]. Date of last access as 16-03-2026. https://aws.amazon.com/q/developer/

GitHub – GitHub Copilot: Your AI Pair Programmer – GitHub Documentation. [online resource]. Date of last access as 16-03-2026. https://github.com/features/copilot

Google Cloud – Codey: Generative AI for Developers – Google Cloud Documentation. [online resource]. Date of last access as 16-03-2026. https://cloud.google.com/vertex-ai/docs/generative-ai/code

ISO/IEC – ISO/IEC 23894: Information Technology – Artificial Intelligence – Risk Management – International Standard. [online resource]. Date of last access as 16-03-2026.

Microsoft – GitHub Copilot for Business and Enterprise – Microsoft Learn. [online resource]. Date of last access as 16-03-2026. https://learn.microsoft.com/github/copilot/copilot-for-business

OpenAI – Codex: An AI System for Translating Natural Language to Code – OpenAI Blog. [online resource]. Date of last access as 16-03-2026. https://openai.com/index/openai-codex/