The Impact of AI Coding Assistants on Software Engineering: A Longitudinal Study

ANNIE VELLA, University of Auckland, New Zealand
KELLY BLINCOE, University of Auckland, New Zealand

AI coding assistants have become prolific in recent years. Through a longitudinal mixed-methods investigation, we examined how professional software engineers perceive the effects of AI coding assistants in regard to task focus, developer experience, and productivity. Two questionnaires were administered six months apart, yielding 158 eligible participants at the first time point, 101 at the second, and a matched longitudinal cohort of 95. Participants reported spending less time on most development tasks, with 82% reporting less on writing code. We found a broader shift in focus from creation to verification activities. We propose a new category of work we term *supervisory engineering work*, encompassing the direction, evaluation, and correction of AI output. This construct is broader than the extra review overhead that has been previously reported in literature since it encompasses directing and shaping AI output both before and during creation. We also identified a productivity-experience paradox: consistently positive productivity perceptions alongside eroding developer experience. These findings suggest that AI coding assistants may be reshaping both the nature of software engineering work and how engineers experience it.

CCS Concepts: • **Software and its engineering → Software development techniques**; • **Human-centered computing → Empirical studies in collaborative and social computing**.

Additional Key Words and Phrases: AI Impact, Software Engineering, Supervisory Engineering, Productivity-Experience Paradox

ACM Reference Format:
Annie Vella and Kelly Blincoe. 2026. The Impact of AI Coding Assistants on Software Engineering: A Longitudinal Study. *ACM Trans. Softw. Eng. Methodol.* 1, 1 (January 2026), 33 pages. https://doi.org/10.1145/3861343

1 Introduction

AI coding assistants are changing the way software engineers work. Tools such as GitHub Copilot and Cursor, along with general-purpose AI systems like ChatGPT and Claude, have become embedded in everyday development work in just a few years [52, 53]. Promising reports of productivity gains [15, 42] have driven rapid adoption, both by organisations seeking efficiency and by individuals seeking support across a wide range of development tasks. Yet, the 2025 Stack Overflow survey found that while 60% of developers view AI tools favourably, this represents a decline from over 70% in previous years [53]. The same survey found that more developers now distrust the accuracy of AI output than trust it, and that a majority report frustration with "AI solutions that are almost right, but not quite" [53]. This mixed sentiment suggests that widespread use coexists with reservations. Beyond productivity, these tools may be reshaping where engineers direct their effort and how they experience their work.

Much of the early research in this area has centered on evaluating the technical capabilities of AI coding assistants: the correctness, security, and maintainability of generated code [35, 38, 43, 62]. Other studies have examined how

Authors' Contact Information: Annie Vella, annie.luxton@gmail.com, University of Auckland, Auckland, New Zealand; Kelly Blincoe, k.blincoe@auckland.ac.nz, University of Auckland, Auckland, New Zealand.



This work is licensed under a Creative Commons Attribution 4.0 International License.
© 2026 Copyright held by the owner/author(s).
Manuscript submitted to ACM

engineers use these tools in practice [36, 40, 48], several reporting increased review effort and a shift from producing code toward assessing it [7, 56]. These accounts are snapshots, however, leaving open how the composition of engineers' work evolves with sustained use. A smaller body of work explores developers' perceptions, covering trust, usability, and impacts on learning [27, 30], though often with students rather than professional practitioners [57]. While productivity has been the focus of much research [22, 60], developer experience has received less attention despite being identified as a driver of productivity [32, 37, 54], and it is unclear whether that relationship holds when AI is embedded in development workflows. Further, what remains largely missing is longitudinal evidence of how professional software engineers experience these tools, and of how those experiences change.

We address this gap through a longitudinal mixed-methods study of professional software engineers using AI coding assistants. Participants completed two questionnaires six months apart, with 158 eligible respondents initially, 101 at follow-up, and 95 completing both phases. By prioritising lived experience over tool performance, this study offers a practice-oriented view of how AI coding assistants are influencing software engineering work. We examine perceptions of how focus shifts across tasks and perceptions of productivity and developer experience, guided by two research questions:

- **RQ1:** How does the introduction of AI coding assistants shift software engineers' perceived focus across various development tasks, such as designing, writing, refactoring, testing, debugging, and reviewing code?
- **RQ2:** How does the use of AI coding assistants impact software engineers' perceptions of their developer experience and productivity?

The main contributions of this study are:

- **(RQ1)** Evidence of a *creation-to-verification shift*: most participants perceived spending less time on the development tasks we measured, with writing code showing the steepest decline. The balance tilted modestly toward verification activities over the study period.
- **(RQ1)** A proposed new category of development work, *supervisory engineering work*, encompassing the effort required to direct AI, evaluate its output, and correct its errors. This construct, which emerged from convergent quantitative and qualitative patterns and encompasses more than just additional reviewing overhead, may help explain where perceived time savings from traditional tasks are being reallocated.
- **(RQ2)** Evidence of a *productivity-experience paradox*: consistently positive productivity perceptions alongside eroding developer experience, especially in flow state and cognitive load. This pattern suggests the established relationship between developer experience and productivity may operate differently when AI is embedded in workflows, and how we define and measure developer experience may need to evolve.

Together, these findings offer a grounded perspective on how AI coding assistants may be transforming professional practice, and what that could signal for the future of software engineering.

2 Literature Review

This section examines existing research on the impact of AI coding assistants, spanning controlled experiments measuring productivity gains, surveys capturing developer perceptions, and studies examining workflow changes.

2.1 Productivity and Workflow

Developer productivity has been a primary focus of AI coding assistant research. A number of controlled experiments found that task completion time reduced significantly with the use of AI [42, 59], as did an A/B test within a large

bank [12] and a task-level evaluation on proprietary codebases [40]. Randomised field experiments running over several months measured completed tasks rather than task completion time, finding increases overall, with the largest gains among less experienced developers [15]. Perceptions on productivity remain broadly positive in industry surveys [19, 22, 52]. Though, a study at IBM found that while productivity increased for most, some developers saw no gains at all, suggesting gains may not be uniform [60]. Another study employing a randomised controlled trial found that AI tool usage had actually *increased* task completion time by 19% even though developers estimated AI had reduced their completion time by 20% [6]. This finding is notable but difficult to generalise due to the small sample and heterogeneous tasks.

Research has found that developers do not always complete tasks faster, due to more time spent crafting prompts and understanding, editing, and debugging AI output [36, 57]. Contributing to this, developers report difficulty steering the tools toward the desired output and that outputs are not always correct [17, 30]. Studies report increased review effort and describe a role shift from direct production toward assessing and integrating suggestions [7, 56]. Barke et al.'s grounded theory of Copilot use distinguishes acceleration, where the developer knows what they want next, from exploration, where they are unsure and validation is more extensive [5]. Banh et al. describe an underestimated overhead in organisational adoption, arising from correcting hallucinated output, the complexity of prompting, and the cost of deployment, and report that productivity gains may be diminished by the need for manual correction [4]. At a broader level of task allocation, a study found that access to GitHub Copilot shifted effort toward core coding activities and away from peripheral project management tasks, with effects strongest among less experienced developers [21]. These studies suggest that effort reallocation is occurring.

Research sheds some light on the reasons for these shifts. First, what developers delegate depends on what they enjoy: writing tests and documentation were rated least enjoyable and most likely to be delegated to AI, while implementing new features was most enjoyable and least likely to be delegated [48]. This aligns with broader findings that developers want AI to reduce toil, freeing them to focus on complex problem-solving [16]. Creativity in software engineering, often expressed through refactoring, designing reusable code, and learning, is something developers want to preserve [24]. Second, capability varies with task complexity and context: benchmarks show progress on isolated problems, but performance drops on multi-file, real-world issues and varies by language and difficulty [25, 27, 38, 62].

These studies indicate that AI coding assistants redistribute engineers' focus and effort, though questions remain open. Controlled studies capture behaviour on predefined tasks, but how these patterns manifest in the varied contexts of day-to-day professional work is less well understood. Furthermore, both the underlying AI models and the tools built on them are evolving rapidly as adoption continues to grow. Understanding how engineers' focus shifts as these tools mature, and how engineers perceive those shifts, would benefit from research that tracks practitioners over time.

2.2 Developer Experience

The Developer Experience (DevEx) framework identifies three dimensions: feedback loops (the speed at which developers receive information about their work), cognitive load (the mental effort required to complete tasks), and flow state (the state of full immersion and enjoyment) [39]. Prior research has shown that developer experience improves productivity, satisfaction, and retention [32, 37, 54]. This evidence base predates the widespread professional use of AI coding assistants. Whether the three dimensions remain the right ones, and whether the relationship they have with productivity still holds, is an open question in AI-assisted contexts. Studies report higher satisfaction, engagement, and enjoyment when coding with AI assistance, yet concerns exist about reduced autonomy, decreased motivation,

or persistent uncertainty about correctness [11, 13]. These trade-offs suggest that productivity gains may not always translate into improved experience.

While some research has investigated each of the DevEx dimensions in relation to AI coding assistants, the results are not conclusive. AI coding assistants may shorten *feedback loops* by providing immediate suggestions, reducing pull request review cycle time, and reducing reliance on colleagues [22, 28]. However, the usefulness of this feedback depends on its quality and relevance, and developers still report needing to validate AI output carefully [17].

Some studies found that *cognitive load* was reduced from having information centralised within the IDE and eliminating context-switching to external sources [44]. However, evaluating AI outputs can also increase cognitive demands, particularly for complex tasks or when suggestions are overly verbose and lack project-specific context [1]. Another study found that the opacity and unpredictability of AI-generated code led to elevated stress markers and frequent interpretive confusion among novice users [63]. These findings suggest that cognitive demands may not simply decrease with AI assistance; they shift and redistribute. How AI coding assistants affect *flow state* is not yet well understood. We found only one interview study that examined this; the findings suggest that whether AI disrupts or supports flow often depends on the task at hand rather than the tool alone [29].

Overall, prior work indicates that AI coding assistants can improve aspects of developer experience while simultaneously introducing new sources of cognitive effort and fragmentation.

2.3 Gaps in Current Understanding

The most significant gap is temporal. Much of the evidence described above relates to a single point in time: a survey covering five development activities [48], single-session observation [5], and one interview per participant [4], a limitation the last two acknowledge themselves. Controlled experiments are similarly restricted, measuring task performance within one session [42, 57].

Some evidence does extend over time, but not far enough, or not in the same people. A six-week trial at a large bank surveyed participants repeatedly as they adopted an assistant [12], though a structured evaluation of that length focuses on early adoption rather than sustained use. A three-week randomised trial compared developers' beliefs before and after they gained access to an assistant, and its authors recommend that future work examine longer durations, of six months to a year, to assess whether further changes emerge [11]. The Stack Overflow developer survey runs annually and reported sentiment shifting considerably between 2024 and 2025 [52, 53], which suggests perceptions do evolve with familiarity, but since each year samples a different population, it does not help us understand how an individual developer's views change over time. A systematic review of 39 studies on LLM assistants and developer productivity also suggests that most research in this space lacks longitudinal designs [33].

A second gap relates to population. Evidence on cognitive load under AI assistance comes from novice programming students [23], as does the closest study of usage and perceptions during a software engineering project [46]. Studies of novice programmers often rely on educational populations [45]. Professional software engineering generally involves different pressures: legacy codebases, ambiguous requirements, team coordination, and real consequences for failure. Whether findings from educational contexts can be applied to professional settings is questionable.

A third gap concerns what is measured. Many studies measure what AI tools achieve rather than how engineers experience working with them, including those that track developers over time. Randomised field experiments have followed developers for several months while measuring completed tasks [15]; analyses of repository activity collect no self-reports from the developers they study [21, 61]; and studies combining long-run activity data with a developer survey administer that survey once [49, 55]. Interview and observational studies have characterised how developers

interact with these tools [5] and how generative AI is adopted in practice [4], rather than how the balance of activities in the work itself changes. Within experiential research, feedback loops and flow state have received much less direct attention than cognitive load. Where they appear, feedback loops are inferred from process metrics such as review cycle time [28], and flow state captured as one item within a broader productivity score [64], rather than studied as experiential dimensions in their own right. Productivity and developer experience are also rarely measured together in the context of AI, yet the two likely interact: an engineer who perceives productivity gains but experiences cognitive strain may develop different attitudes than one who experiences both benefits. An enterprise study of one coding assistant comes closest, pairing a large-scale survey with usability testing to examine perceptions of productivity alongside the user experience of the assistant [60], but it captures a single snapshot and does not examine feedback loops, cognitive load, or flow state.

These gaps motivate the research questions that guide this study. By tracking a cohort of professional software engineers over six months, we address the temporal gap directly. By focusing on practitioners rather than students, we address the population gap. By examining perceptions across task focus (RQ1) and developer experience and productivity (RQ2), we address the gap in what is measured.

3 Methodology

This section describes the design, data collection, and analysis approach used to investigate how professional software engineers perceive the impact of AI coding assistants.

3.1 Study Design

We adopted a pragmatic interpretivist stance for this research due to the socio-technical nature of the topic and the student researcher's position as a practising software engineer.

3.1.1 Longitudinal Design. We invited the same participants to complete two questionnaires six months apart. AI coding assistants are evolving rapidly and adoption continues to grow; a single time point would only have captured a snapshot, whereas two time points allowed us to examine how perceptions evolved over time. The first questionnaire (Q1) was administered in October 2024 and the second (Q2) in April 2025, each open for four weeks.

3.1.2 Data Collection Approach. Both questionnaires were administered online through Qualtrics, enabling us to reach professional software engineers globally. Each was designed to take approximately 15 minutes to complete, and followed a convergent parallel mixed-methods design [14], with qualitative accounts contextualising quantitative patterns.

3.1.3 Questionnaire Structure. Each questionnaire had seven sections, of which four are aligned to our research questions in this paper: qualifying questions; experience and perceptions of AI tools; perceived productivity and developer experience; participant demographics. The survey also included questions on perceived impacts on skill development and self-efficacy, prompt engineering, and broader reflections on the profession; this data is not reported in this publication. The questionnaires included Likert-scale, multiple choice, ranking, and open-ended questions to support a mixed-methods analysis. The full study instrument is provided in our online replication package [58].

The *Qualifying Questions* section at the beginning of both questionnaires was used to check our eligibility criteria. Participants were required to be professional software engineers and to be currently using AI coding assistants. Those who answered "No" to either qualifying question were routed to the end of the questionnaire and did not complete the remaining sections.

The *Experience and Perceptions of AI Tools* section captured tool usage patterns and attitudes, both providing study context as well as helping to answer RQ1. To provide context of AI use and perceptions, we asked participants which AI coding assistants they used, how frequently, and their initial impressions (in Q1) or anticipated disappointment if tools were removed (in Q2). Participants also ranked seven concerns about using AI coding assistants in their workflow: quality, readability, maintainability, security, dependency, learning curve, and other; we use *primary concern* to refer to whichever category a participant ranked first.

To answer *RQ1* (How does the introduction of AI coding assistants shift software engineers' perceived focus across various development tasks?), the section also asked how participants perceived changes in focus across six core development tasks: designing, writing code, refactoring code, reviewing code, testing, and debugging. These tasks reflect core activities identified in empirical studies of software engineering practice [31]. Participants were asked "When using an AI coding assistant, do you feel you spend more or less time on the following software engineering tasks?" and rated each task on a five-point scale from "Much less" to "Much more", with "About the same" at the midpoint. The tasks were presented as labels without further definition, so participants interpreted them in the context of their own work. Participants were also asked to describe how AI coding assistants had changed their workflow.

The *Perceived Productivity and Developer Experience* section captured data to answer *RQ2* (How does the use of AI coding assistants impact software engineers' perceptions of their developer experience and productivity?). Participants rated their perceived productivity change and the impact of AI coding assistants on each of the three DevEx dimensions [39]: cognitive load, feedback loops, and flow state. They were asked "How has your productivity changed since using an AI coding assistant?" and "How does using an AI coding assistant affect the following aspects of your Developer Experience?", rating each on a five-point scale from "Much worse" to "Much better", with "About the same" at the midpoint. Each of the three dimensions was further defined: feedback loops as "the speed and effectiveness with which you receive and act on feedback during development", cognitive load as "the mental effort required to complete your development tasks", and flow state as "the extent to which you can work seamlessly and maintain concentration without interruptions". Productivity was deliberately not defined, so that participants would apply their own understanding of the term. They were also invited to describe specific instances where AI coding assistants helped or hindered their work in Q1, and any unexpected benefits or challenges they had experienced in Q2.

The *Participant Demographics* section captured gender, age, years of professional experience, self-reported expertise, role, primary programming language, industry, company size, country of residence and country of company headquarters. In Q2, demographic questions were optional; participants could choose to update selected fields (e.g., job role or primary language) if their circumstances had changed since Q1.

3.1.4 Participant Recruitment. We recruited participants using a combination of convenience and referral-chain sampling [3]. The primary recruitment channels were our professional networks, including LinkedIn, X (formerly Twitter), Facebook, Discord, and several Slack communities, where invitations were shared directly or via community administrators. Additionally, 33 organisations were contacted to request internal distribution of which nine agreed to share the invitation with their engineering teams. Recruitment therefore drew from both individual networks and organisational channels, enabling participation from software engineers across a range of roles, experience levels, industries and countries.

3.1.5 Ethics. This study received approval from the University of Auckland Human Participants Ethics Committee (Reference number UAHPEC27902). All participants were provided with a Participant Information Sheet and completed

an electronic consent form. A $200 NZD gift card was offered as a prize draw incentive for those who completed both questionnaires.

3.2 Participants

A total of 224 responses were recorded for Q1. Of these, 35 were incomplete (Progress < 100%), 16 respondents indicated they were not professional software engineers, and 15 reported not using AI coding assistants, yielding a final Q1 sample of 158 eligible participants (71% eligibility rate). All Q1 respondents who completed the questionnaire and confirmed they were professional software engineers were invited to participate in Q2, regardless of whether they reported using AI coding assistants at Q1 ($n = 173$). For Q2, 111 responses were recorded, of which 6 were incomplete and 4 reported not using AI coding assistants, yielding 101 eligible participants (91% eligibility rate).

3.2.1 Longitudinal Attrition Analysis. Of the Q1 and Q2 eligible participants, 95 (60%) participated in both and formed the matched longitudinal cohort. Fisher's exact tests compared those retained versus lost to follow-up across six variables: gender, experience level, role, programming language, company size, and country. No significant differences were found (all $p > 0.05$), and effect sizes were small (Cramér's $V \leq 0.23$), suggesting that attrition was not systematic.

3.2.2 Participant Demographics. Table 1 summarises the participant demographics. Participants represented 28 countries, with the highest participation from New Zealand (41%).

3.3 Data Analysis

3.3.1 Overview. Across both research questions, we followed the same three-step analytical structure.

Step 1: Quantitative Cross-sectional Analysis. For each research question, we first analysed quantitative data from Q1 and Q2 separately. This allowed us to describe baseline perceptions at each time point. Since the Q1 and Q2 samples differ in size ($n = 158$ vs $n = 101$) and are not matched at this step, any difference between them reflects a between-sample comparison rather than within-participant change.

Step 2: Quantitative Longitudinal Analysis. We then restricted analyses to participants who completed both Q1 and Q2. This enabled us to examine within-participant change over the study period.

Step 3: Qualitative Analysis. We conducted a reflexive thematic analysis [8, 9] of the open-ended responses to understand how participants described and contextualised their experiences with AI coding assistants, complementing the quantitative results.

3.3.2 Quantitative Analysis. All quantitative analyses were conducted in R. The packages used are listed in our replication package [58]. All analyses used complete-case data; sample sizes vary slightly across constructs due to occasional missing responses on individual items (missingness was below 1% at both time points).

Measurement and Scoring. Perceived productivity was measured with a single item, task focus with six single items, one for each development task, and developer experience with three single items, one for each dimension. In both multi-item cases, each item was analysed independently rather than combined into a scale. Single items were used to limit respondent burden across two time points, and because each item asks participants for a judgement about their own work rather than combining several indicators to infer something not directly observable. We acknowledge the limitation this imposes: internal-consistency reliability cannot be estimated for a single item. We did test whether

Table 1. Participant Demographics and Organisational Context

Variable	Category	Q1 n (%)	Q2 n (%)
Gender	Man	135 (85%)	86 (85%)
	Woman	19 (12%)	12 (12%)
	Non-binary	1 (1%)	1 (1%)
	Self-described	2 (1%)	1 (1%)
	Not disclosed	1 (1%)	1 (1%)
Age	18–24	5 (3%)	2 (2%)
	25–34	55 (35%)	33 (33%)
	35–44	63 (40%)	45 (45%)
	45–54	28 (18%)	15 (15%)
	55+	5 (3%)	4 (4%)
	Not disclosed	2 (1%)	2 (2%)
Experience Level	Junior	21 (13%)	13 (13%)
	Mid-Career	77 (49%)	51 (50%)
	Senior	59 (37%)	37 (37%)
	Not disclosed	1 (1%)	0 (0%)
Role	Backend	37 (23%)	28 (28%)
	Frontend	10 (6%)	7 (7%)
	Fullstack	57 (36%)	32 (32%)
	Manager/Architect	24 (15%)	16 (16%)
	Mobile	12 (8%)	7 (7%)
	Specialised (e.g., Data)	15 (9%)	9 (9%)
	Other	3 (2%)	2 (2%)
Company Size	Small	44 (28%)	25 (25%)
	Medium	44 (28%)	25 (25%)
	Large	34 (22%)	27 (27%)
	Very Large	33 (21%)	22 (22%)
	Not disclosed	3 (2%)	2 (2%)
Country of Residence	New Zealand	65 (41%)	48 (48%)
	Netherlands	16 (10%)	10 (10%)
	United Kingdom	14 (9%)	11 (11%)
	Australia	14 (9%)	6 (6%)
	United States	9 (6%)	3 (3%)
	Other	40 (25%)	23 (23%)

the three developer experience dimensions could be treated as a composite and rejected it (Cronbach's $\alpha = 0.24$–0.56 across samples and time points), which supports analysing the dimensions separately. Single Likert items are ordinal, so we used non-parametric tests for all comparisons involving these measures. Perceptions were the object of study throughout our investigation, consistent with our pragmatic interpretivist stance. We write *perceived productivity* because "productivity" can also be used to refer to output itself. Developer experience carries no such ambiguity, so we use it without qualification.

Descriptive Statistics. For each outcome, we computed means, standard deviations, and response category distributions for each time point. Likert-scale responses were summarised as numeric scores (1–5) and grouped into meaningful categories (e.g., less/same/more time, positive/negative/neutral). For longitudinal data, we constructed Q1-to-Q2 transition matrices, summarised as movement patterns and visualised using alluvial diagrams.

Inferential Tests. For longitudinal within-participant change, we used paired Wilcoxon signed-rank tests. For effect sizes, we report rank-biserial correlation (r_{rb}).

Correlations. We used Spearman rank correlations due to the ordinal nature of our Likert-scale measures.

Multiple Comparison Corrections. All multiple comparison corrections used the Holm-Bonferroni method. For cross-sectional analyses, correction was applied within each time point separately, as Q1 and Q2 represent independent cross-sections. For constructs measured through multiple related items or dimensions, corrections were applied across all items within that construct: the six task focus items (RQ1) and three developer experience dimensions (RQ2).

3.3.3 Qualitative Analysis. We used reflexive thematic analysis following Braun and Clarke [8, 9] to analyse the open-ended responses. The open-ended questions associated with each research question were analysed across both time points.

Phase 1: Familiarisation with the Data. We read through all open-ended responses across both questionnaires multiple times, actively searching for meanings and patterns rather than passively absorbing content. These initial readings gave us a sense of the diversity of perspectives among professional software engineers and informed preliminary ideas that we carried into the coding phase.

Phase 2: Generating Initial Codes. Using NVivo, we followed an inductive approach, generating codes from participant responses rather than applying a pre-existing framework. Codes were applied across all open-ended questions, as participants frequently discussed similar themes regardless of which question prompted them, making a shared codebook more practical than separate per-question codebooks.

To illustrate the coding process, consider the following response to the question about how AI coding assistants changed workflow:

> *"I find them helpful when working with languages and technologies that I am less familiar with, or use less frequently. They are often wrong but can help you move more quickly; A wrong-but-close answer can give you a jump to the next thing to search for and solve the problem."* (P36, Q1)

Rather than coding the response as a single unit, we identified distinct segments conveying different ideas. The opening clause about working with unfamiliar technologies was coded as *Capability Expansion*; the observation that AI helps "move more quickly" was coded as *Speed & Automation*; and the framing of imperfect output as enabling forward momentum was coded as *Confidence Boost*.

The codebook was iteratively refined throughout the coding process: new codes were added when existing codes did not adequately capture a segment, similar codes were merged to reduce redundancy, and code definitions were clarified to improve consistency. The full codebook is provided in the replication package [58].

For some open-ended questions, we applied an additional layer of cross-cutting valence codes to capture the evaluative dimension of participants' accounts. For example, the responses to the question asking whether AI coding assistants either helped or hindered work, each segment was coded with a valence reflecting the question wording: *helpful* or *hindrance*.

Phases 3–5: Generating, Reviewing, and Defining Themes. Following the coding phase, we developed themes at the research-question level rather than for each open-ended question, examining coded responses from both questionnaires to identify themes spanning the longitudinal period while revealing where accounts differed between time points. Code

frequency summaries helped orient this process, but themes were generated based on conceptual significance rather than prevalence. We reviewed candidate themes against the original responses and refined them through multiple iterative passes until they provided a coherent interpretation of participants' accounts. Some codes contributed to multiple themes depending on contextual framing, and where themes were multi-faceted, sub-themes captured distinct dimensions. Where similar experiences surfaced across multiple research questions, themes were shaped to reflect the specific lens of each question rather than treated as identical.

To illustrate theme generation, for RQ2, codes including *Productivity & Effort Impact*, *Cognitive Load Impact*, and *Prompt Interaction & Refinement* frequently co-occurred in responses describing faster output alongside persistent friction; these were combined into a theme capturing accelerated throughput with uneven day-to-day experience.

Participants worked across a range of roles, industries, organisation sizes and levels of experience, and used different assistants at differing frequencies (Table 1). Demographic data were imported into NVivo alongside the responses, so each segment was read alongside the respondent's stated context rather than as a free-standing statement. That context informed how participant accounts were interpreted. We did not, however, analyse themes by participant subgroup, as the sample could not support claims about differences between groups.

Trustworthiness. All coding was conducted by the first author, a practising software engineer, and that experience shaped interpretation. Reflexive thematic analysis treats this as something to be made explicit and examined rather than eliminated, and we used several reflexivity practices to do so. The first author kept reflexive memos throughout coding and returned periodically to how domain expertise might be influencing interpretation. We actively searched for counter-examples that did not fit developing themes. The second author, who did not code, took part in iterative discussion throughout the analysis rather than at theme review alone: the codebook was reviewed across several rounds of feedback, with attention to whether codes adequately captured the range of participant experiences, and candidate themes were discussed as they developed. Ambiguous segments were flagged during initial coding and revisited after the full dataset was coded to ensure consistent application. Trustworthiness in reflexive thematic analysis derives from this transparency and sustained reflexive engagement rather than from procedural checks such as independent coder agreement.

4 Results

This section presents the findings from both questionnaires. To provide context for the analyses that follow, we begin by describing which AI coding assistants participants reported using, their usage patterns, and their attitudes and concerns about using these tools. We then address each RQ.

4.1 Study Context

4.1.1 AI Coding Assistants in Use. Participants reported using a wide range of AI coding assistants, with 37 unique tools mentioned across both time points (full list in replication package [58]), including 29 at Q1 (n = 158) and 24 at Q2 (n = 101). GitHub Copilot and ChatGPT were the clear front-runners, both at 70% adoption at Q1 and 58% at Q2 (Figure 1). However, many other tools gained in popularity between the two questionnaires. Among matched participants (n = 95), the tools in use evolved considerably over the study period. The number of distinct tools increased from 18 to 24, 82% changed their tool combinations, and the mean number of tools per participant increased from 1.9 (SD = 0.8) to 2.9 (SD = 1.6).

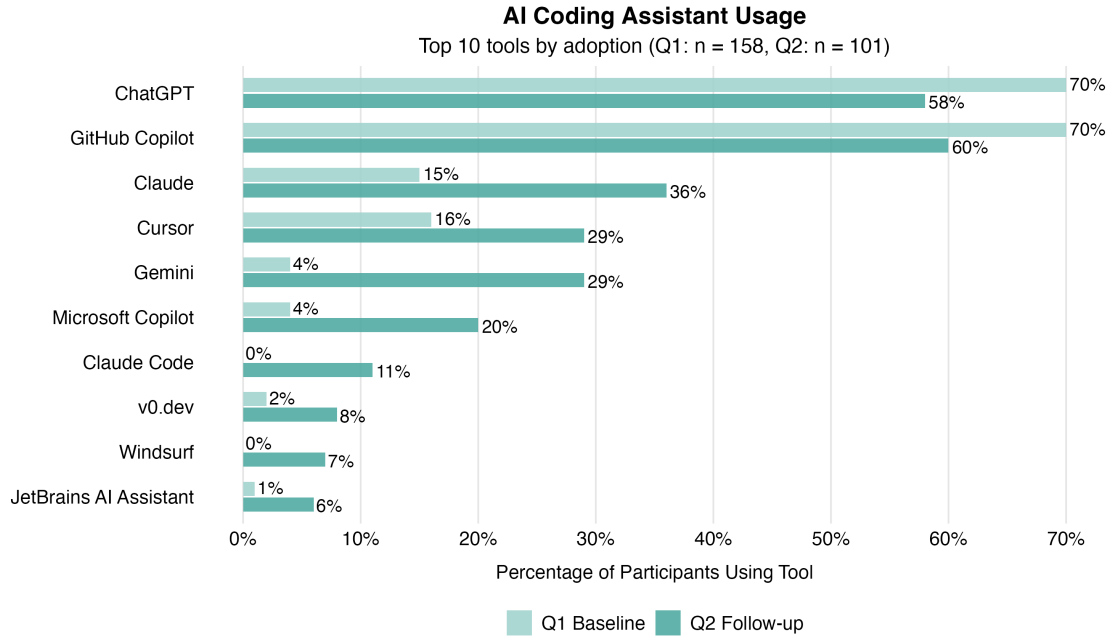


Fig. 1. AI coding assistant adoption rates at Q1 and Q2. Top 10 tools by maximum usage at either time point shown.

4.1.2 Usage Frequency and Location. Over half of the participants in our study used AI coding assistants daily at both time points (57% at Q1 and 61% at Q2). Only a small number reported using AI coding assistants once a week or less (10% at Q1 and 8% at Q2). Usage frequency remained stable across time points (paired Wilcoxon signed-rank, $p = 0.17$, $n = 95$ matched). The majority used them in both work and personal contexts (65% at Q1, 70% at Q2), with work-only usage less common (27% at Q1, 24% at Q2) and personal-only usage rare (8% at Q1, 6% at Q2).

4.1.3 Attitudes Towards AI Coding Assistants. Participants' attitudes towards AI coding assistants were captured through two measures: initial impressions at Q1 (Figure 2) and anticipated disappointment if tools were removed at Q2 (Figure 3). Initial impressions were predominantly positive, while anticipated disappointment was more evenly distributed. Among matched participants, the two measures showed a moderate positive correlation ($\rho = 0.30$, $p = 0.003$).

4.1.4 Concerns About Using AI Coding Assistants. Participants ranked seven concerns about AI coding assistants from most to least concerning. Quality was the dominant primary concern at both time points, though it declined from 44% to 36%, followed by security. The most notable shift was in maintainability, which rose from 3% to 19% as a primary concern between Q1 and Q2 (Figure 4). Paired Wilcoxon signed-rank tests confirmed maintainability was the only statistically significant change in ranking ($n = 89$ matched, $p = 0.003$, moderate effect).

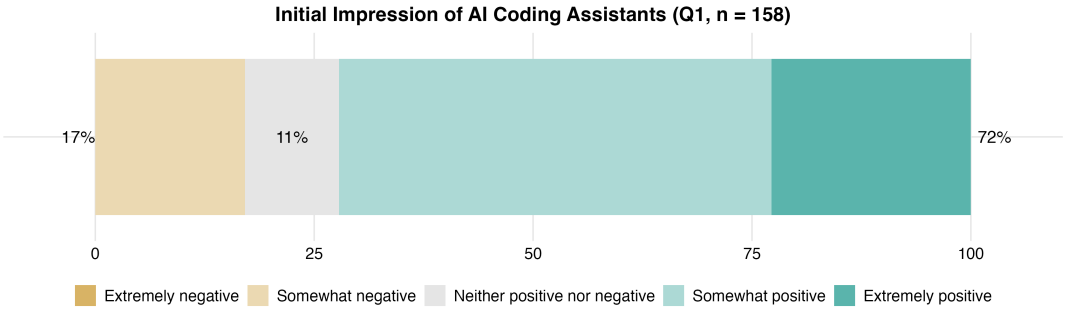


Fig. 2. Distribution of initial impressions of AI coding assistants at Q1.

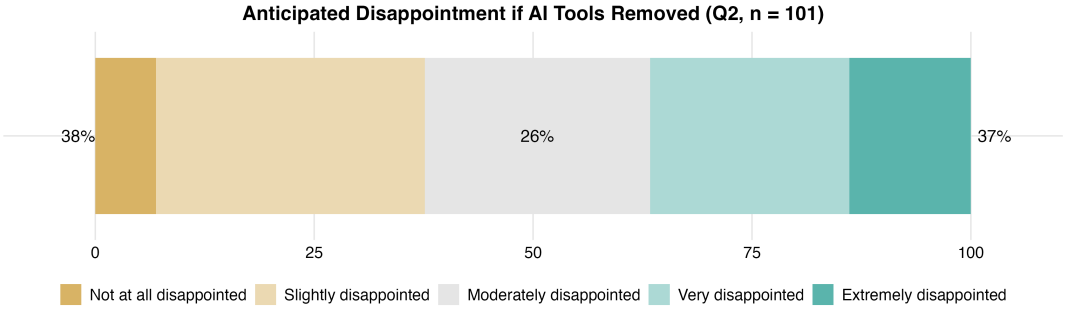


Fig. 3. Distribution of anticipated disappointment if AI coding assistants were removed at Q2.

4.2 RQ1: Task Focus Shifts

To examine how the introduction of AI coding assistants shifts perceived task focus, we analyse perceived time reallocation across six core development activities: designing, writing, refactoring, testing, debugging, and reviewing code.

4.2.1 Cross-sectional Analysis. Perceived time fell across most development tasks, with writing code furthest below neutral and reviewing code the only task above it (Figure 5). Writing code showed the strongest reduction at both time points ($M = 2.10$ at Q1, $M = 1.93$ at Q2), with 82% of participants reporting less time by Q2 and only 2% reporting more. Refactoring code and testing also sat below neutral, while designing and debugging were near neutral at both time points. Reviewing code was above the midpoint at both (Q1: 3.03, Q2: 3.15).

4.2.2 Longitudinal Change in Task Focus. Next, we examined how task focus perceptions changed within participants ($n = 88$ matched participants). Paired Wilcoxon signed-rank tests with Holm-Bonferroni correction showed no statistically significant changes, though effect sizes varied from very small to moderate (Table 2). Writing code showed the largest effect in the expected direction ($r_{rb} = -0.35$, moderate effect), while testing and reviewing showed small increases ($r_{rb} = 0.29$ and $r_{rb} = 0.26$ respectively).

Table 2. Longitudinal Wilcoxon Signed-Rank Tests: Changes in Task Focus Perceptions Within Matched Participants (n = 88)

Task	Q1		Q2			Test Statistics		
	M	SD	M	SD	ΔM	V	p_{adj}	r_{rb}
Designing	2.94	0.72	2.89	0.94	-0.06	595.5	1.000	-0.10
Writing code	2.10	0.80	1.92	0.76	-0.18	255.0	0.263	-0.35
Refactoring code	2.48	0.93	2.39	1.06	-0.09	688.0	1.000	-0.11
Reviewing code	2.93	0.83	3.16	1.04	+0.23	967.0	0.333	0.26
Testing	2.52	0.92	2.77	0.98	+0.25	1030.5	0.263	0.29
Debugging	2.84	0.93	2.84	0.79	0.00	556.0	1.000	-0.01

Task focus was rated on a five-point scale: 1 = much less time, 3 = about the same, 5 = much more time spent on the task when using an AI coding assistant.
ΔM = Mean change (Q2 - Q1). V = Wilcoxon signed-rank test statistic. r_{rb} = rank-biserial correlation.

While aggregate statistics showed minimal change, individual transition patterns revealed substantial variation beneath these stable averages. Here we examine the three tasks with the largest effect sizes: writing code (negative) and testing and reviewing code (positive).

Table 3 confirms divergent patterns: writing code showed high stability (56%) with continued downward drift (30% moved toward less time), while testing and reviewing showed upward movement (42% and 40% toward more time). Only 8% both reduced writing time and increased reviewing time between time points. This is a constraint of the measure

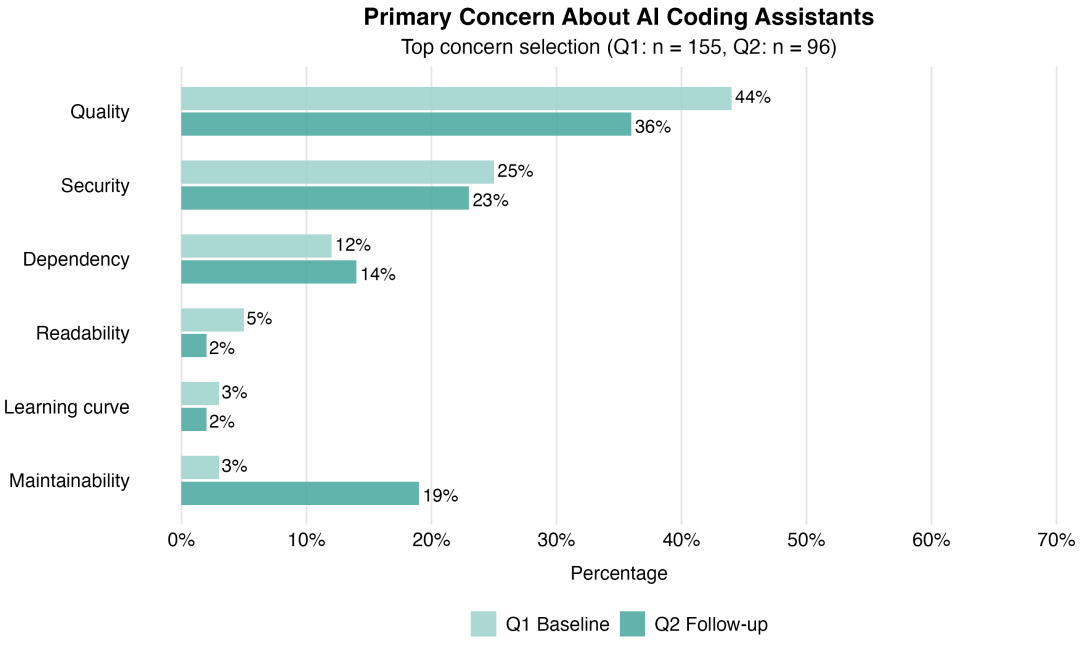


Fig. 4. Primary concern distribution at Q1 and Q2. "Other" excluded from analysis.

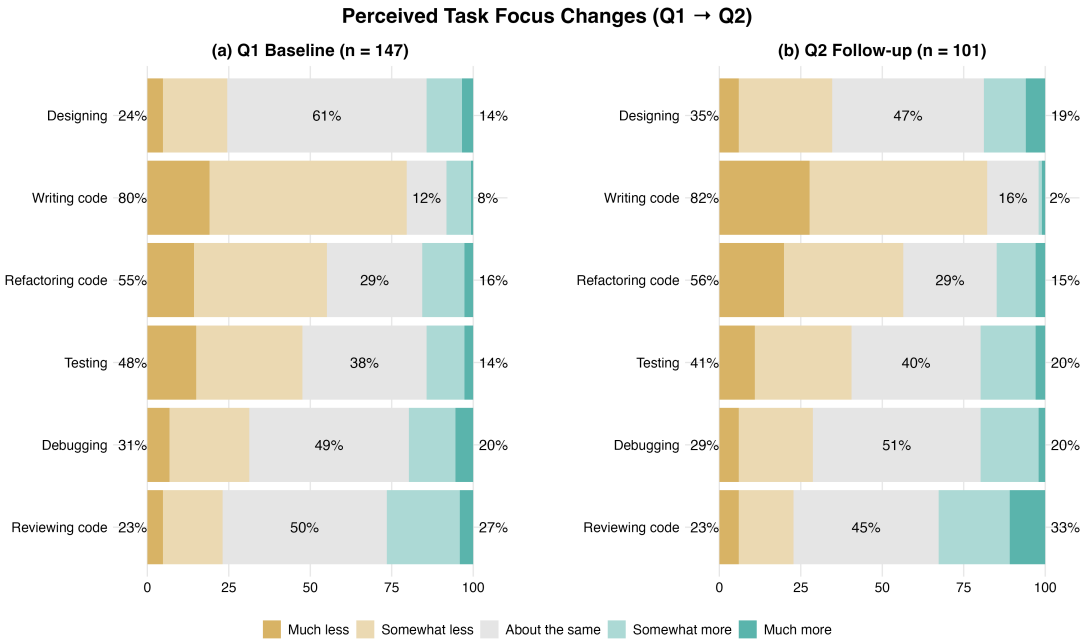


Fig. 5. Task focus shifts across time points.

Table 3. Transition Patterns for Key Tasks ($n = 88$)

Task	Less	Stable	More
Writing code	30% ($n = 26$)	56% ($n = 49$)	15% ($n = 13$)
Testing	22% ($n = 19$)	36% ($n = 32$)	42% ($n = 37$)
Reviewing	23% ($n = 20$)	38% ($n = 33$)	40% ($n = 35$)

Each participant's Q1 and Q2 ratings were compared. *Less* and *More* indicate that the rating moved down or up between time points; *Stable* indicates the same rating at both. *Stable* therefore means that the participant's assessment was unchanged, not that their time on the task was unchanged. Percentages may not sum to 100% due to rounding.

rather than a behavioural pattern: 81% had already reported spending less time writing code at Q1, leaving little room for a further reduction to register.

Creation-to-Verification Shift Analysis: Beyond the individual tasks, we examined whether the balance between creation- and verification-oriented work shifted. Following the Vee model [34], we grouped the six tasks into *creation tasks* (designing, writing, refactoring) and *verification tasks* (reviewing, testing, debugging). These are theoretical distinctions rather than psychometric subscales, so we retain them despite low internal consistency (Creation: $\alpha = 0.40$–0.49; Verification: $\alpha = 0.56$–0.57). Table 4 presents the results.

Neither creation nor verification tasks changed significantly in isolation ($p_{adj} = 0.092$ and 0.060 respectively). However, the balance between task groups showed a significant shift toward verification ($p_{adj} = 0.006$, $r_{rb} = 0.39$, moderate effect), with the mean balance score increasing from 0.26 to 0.53. When examining individual participant shifts, 42% showed stability on at least one dimension, with the remaining 58% shifting. Of these, decreased creation

and increased verification was the most common shift (20%), followed by both dimensions increasing (17%) and both decreasing (15%).

Table 4. Task Group-Level Wilcoxon Signed-Rank Tests: Creation vs Verification Task Group Averages (n = 88)

Task Group	Q1		Q2			Test Statistics		
	M	SD	M	SD	ΔM	V	p_{adj}	r_{rb}
Creation	2.51	0.58	2.40	0.64	-0.11	814.5	0.092	-0.24
Verification	2.77	0.66	2.92	0.69	+0.16	1527.5	0.060	0.30
Balance (V-C)	0.26	0.58	0.53	0.83	+0.27	2365.5	0.006	**0.39****

Each group score is the mean of its three constituent task scores.
ΔM = Mean change (Q2 - Q1). Balance = Verification - Creation. V = Wilcoxon signed-rank test statistic. r_{rb} = rank-biserial correlation.
Significance: $^{*}p < 0.05$, $^{**}p < 0.01$, $^{***}p < 0.001$. Holm-Bonferroni corrected.

4.2.3 Qualitative Perspectives on Workflow Changes. Reflexive thematic analysis of responses describing how AI coding assistants had changed participants' workflows, identified four themes capturing changes in their interactions with code, tools, and information sources.

Theme 1: Compressing routine creation work. Participants across both time points consistently described AI coding assistants as reducing the effort required for routine implementation tasks. Their accounts locate where the perceived reduction in writing code came from: boilerplate, repetitive structures and small functions.

Sub-theme 1A: Reducing manual implementation effort. Participants at both time points highlighted how AI tools helped them generate boilerplate, small functions, or repetitive structures. As one participant explained, *"I spend less time writing repetitive and boilerplate code"* (P25, Q1) and *"Mostly, the routine tasks like coding and testing now require less manual effort."* (P98, Q2). Others noted that they now typed less: *"Reduced my keystrokes. I mostly use AI only for line completion and most of the time I only have to type less than half a line before AI figures out what I want"* (P44, Q1).

Sub-theme 1B: Supporting small-scale debugging and error handling. Participants described targeted improvements in resolving smaller issues, such as obtaining explanations of error messages or addressing minor test failures. This pattern appeared consistently across both questionnaires. As one participant noted, *"I give the error message directly to ChatGPT and get a primer on what could be the possible cause of the issue"* (P135, Q1). Another wrote, *"Troubleshooting and debugging become faster."* (P91, Q1).

Theme 2: Rerouting information seeking and comprehension into AI interactions. Participants described noticeable changes in how they acquired information, understood unfamiliar code, and learned new concepts. This theme appeared strongly at Q1 and remained clear at Q2, although Q2 participants often described these behaviours as normalised rather than novel. Information seeking was not one of the six rated activities, so this change sits outside the task measures.

Sub-theme 2A: From search engines to real-time assistance. Developers increasingly used AI as the first step when encountering errors, clarifying APIs, or checking implementation details, for example: *"Instead of going to Stack Overflow for a coding problem, I now go to ChatGPT first, even before any search engine."* (P135, Q1) and *"Instead of googling the syntax for one language, I immediately ask ChatGPT to provide an example"* (P149, Q2). For many, AI effectively displaced older search habits: *"It has basically replaced google for me"* (P143, Q1). By Q2 this was described as settled practice: *"AI coding Assistants are my first source of information."* (P174, Q2).

Sub-theme 2B: Accelerated learning, onboarding, and code understanding. Participants in both time points used AI to accelerate understanding across unfamiliar languages, frameworks, or legacy codebases. For example, one participant wrote: *"Much easier to pick up new tools, libraries, languages and get productive."* (P132, Q1). Another wrote: *"It has been particularly helpful in explaining legacy code and detecting bugs more efficiently."* (P2, Q2). Others reported gaining conceptual clarity: *"Definitely felt like it's given me a level up on understanding new concepts"* (P5, Q2).

Theme 3: From doing to supervising AI-generated code. Across both time points, engineers described shifting effort from directly producing code to supervising AI-generated output. For example, *"I ask AI to make changes first and review them before accepting or rejecting them"* (P82, Q1). In the quantitative results this appears as a shift in the creation-to-verification balance rather than as an increase in reviewing code.

While some participants framed this shift in terms of "design" or "higher-level thinking", their descriptions more consistently reflected supervision-oriented activities, such as directing the assistant, evaluating its suggestions, and deciding what to accept, modify, or discard. This supervisory work replaced portions of hands-on implementation effort but does not map cleanly onto traditional software development task categories. As one participant explained, *"Sometimes, I'll not be happy with the original reference, and will 'coach' the AI to produce something that works better for my use-case"* (P75, Q1). At Q2, this supervisory framing became slightly more prominent. Participants increasingly articulated the effort involved in directing and controlling the assistant itself. One described their role as *"Mostly reading the code and directing AI on the right way"* (P54, Q2).

Theme 4: Verification and trust calibration moderating perceived benefits. Participants varied in how much they trusted or relied upon AI-generated code. Some reported substantial workflow improvements, while others emphasised the need to verify every suggestion or restrict use to low-risk contexts. This theme was evident at both time points. One participant described this careful approach succinctly: *"I almost never let it write production code, or at least never leave in code that I don't understand and or wouldn't have written myself"* (P119, Q1). Others stressed the limits of trust, particularly for complex or critical work: *"I do not trust it for strategic or critical code."* (P8, Q1).

At Q2, similar concerns remained evident, though participants reflected on how their use of AI had evolved over time. One participant noted, *"I think 6 months ago, I was using the tools a lot more cautiously with a lot more fact checking"* (P109, Q2), while another highlighted the increasing verification effort required: *"has increased the 'is this code correct', 'is it maintainable', 'is it secure' and 'is this just probabilistic hallucination' time up."* (P88, Q2). The Q2 accounts moved in both directions: some participants described becoming less cautious over time, while others described a growing verification burden.

4.2.4 Summary. A majority of participants (69%) perceived spending less time overall on the six development tasks at both time points. Writing code showed the largest reduction, with 82% reporting less time by Q2 and means well below neutral at both time points ($M = 2.10$ at Q1; $M = 1.93$ at Q2). All tasks except reviewing showed means below neutral, with writing code furthest from the midpoint. Testing and reviewing were the only tasks with positive effect sizes, though neither change was statistically significant. Among matched participants, a significant shift toward verification activities emerged ($r_{rb} = 0.39$, $p_{adj} = 0.006$, moderate effect). Participants' accounts contextualise this quantitative pattern by describing compressed routine implementation, rerouted information seeking, a new supervisory layer of directing and evaluating AI output, and ongoing verification and trust calibration that moderated perceived benefits.

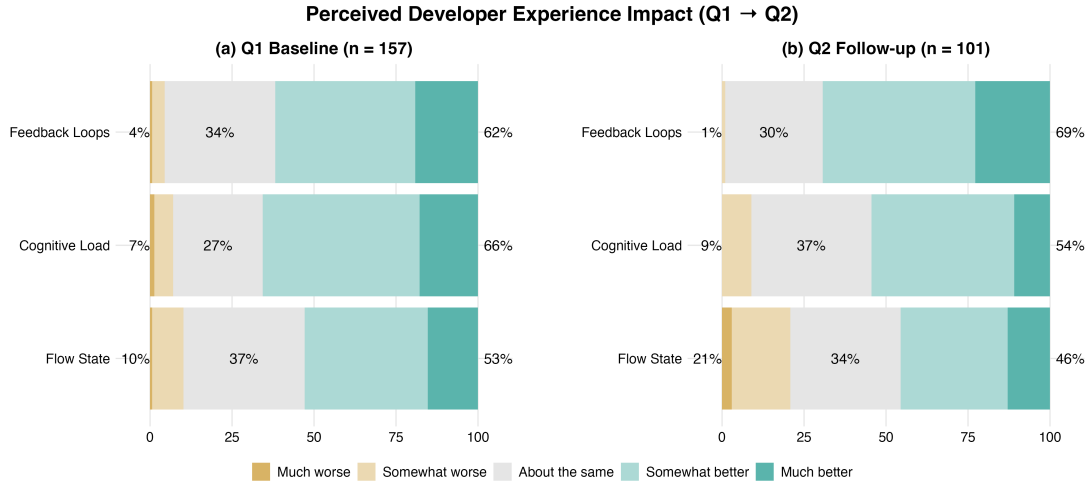


Fig. 6. Perceived impact on developer experience across time points.

4.3 RQ2: Developer Experience and Productivity

To answer RQ2, we analyse two related constructs: developer experience (cognitive load, feedback loops, and flow state) and perceived productivity.

4.3.1 Developer Experience.

Cross-sectional analysis. The three developer experience dimensions were closely grouped at Q1 but more widely separated at Q2, with feedback loops rated highest and flow state lowest (Figure 6). At Q1, participants reported similar improvements in cognitive load ($M = 3.75$) and feedback loops ($M = 3.76$), while flow state lagged behind ($M = 3.57$) and attracted more negative ratings. By Q2, feedback loops showed the greatest improvement ($M = 3.91$) while cognitive load ($M = 3.56$) and flow state ($M = 3.35$) showed smaller improvements. One-sample Wilcoxon signed-rank tests showed that all three dimensions were rated above the neutral midpoint at both time points ($p_{adj} < 0.01$), with effect sizes ranging from moderate (flow state at Q2: $r_{rb} = 0.43$) to large (feedback loops at Q2: $r_{rb} = 0.98$).

We also examined relationships between developer experience and attitudes towards AI tools, since attitudes may shape how developers use these tools and perceive their impact. Table 5 shows Spearman correlations between attitudes towards AI tools and developer experience dimensions. Both initial impressions at Q1 and anticipated disappointment at Q2 showed significant positive associations with all three dimensions, with feedback loops consistently showing the strongest association (moderate effect at both time points). Positive correlations indicate more positive attitudes are associated with better developer experience.

Longitudinal Change in Developer Experience. Next, we examined how developer experience perceptions changed within participants ($n = 94$ matched participants). Paired Wilcoxon signed-rank tests with Holm-Bonferroni correction revealed divergent trajectories across dimensions (Table 6). Only feedback loops improved significantly; cognitive load and flow state showed non-significant declines. Individual variation was substantial: for feedback loops, 33% improved and 18% declined; for cognitive load, 19% improved and 29% declined; for flow state, 27% improved and 35% declined.

Table 5. Spearman Correlations Between Attitudes and Developer Experience Dimensions

Dimension	Q1 (n = 157) Initial Impressions		Q2 (n = 101) Anticipated Disappointment	
	ρ	p_{adj}	ρ	p_{adj}
Feedback Loops	**0.32*****	<0.001	**0.33****	0.002
Flow State	**0.27****	0.001	**0.26***	0.015
Cognitive Load	**0.22****	0.007	**0.23***	0.021

$^{*}p < 0.05$, $^{**}p < 0.01$, $^{***}p < 0.001$ (Holm-Bonferroni corrected)

Table 6. Longitudinal Wilcoxon Signed-Rank Tests: Changes in Developer Experience Dimensions Within Matched Participants (n = 94)

Dimension	Q1		Q2			Test Statistics		
	M	SD	M	SD	ΔM	V	p_{adj}	r_{rb}
Feedback Loops	3.73	0.78	3.95	0.74	+0.21	810.5	0.038	**0.38***
Cognitive Load	3.74	0.80	3.60	0.79	−0.15	382.5	0.195	−0.26
Flow State	3.54	0.84	3.36	1.01	−0.18	651.5	0.195	−0.24

Developer experience was rated on a five-point scale: 1 = much worse, 3 = about the same, 5 = much better, describing how using an AI coding assistant affects each dimension.
ΔM = Mean change (Q2 - Q1). V = Wilcoxon signed-rank test statistic. r_{rb} = rank-biserial correlation.
Significance: $^{*}p < 0.05$, $^{**}p < 0.01$, $^{***}p < 0.001$.

Among matched participants, we classified perceptions into cohorts based on meaningful thresholds: *Positive* (all three dimensions rated 4–5), *Negative* (any dimension rated 1–2), and *Mixed/Neutral* (neither fully positive nor containing any negative ratings). Figure 7 visualises how participants transitioned between these cohorts over the study period. The cohort analysis reveals a drift toward negative developer experience. The Negative cohort nearly doubled from 14% to 27%, while the Positive cohort showed poor retention at 37%. No participant who started in the Negative cohort recovered to fully Positive by Q2. Complete stability across all three dimensions was rare (9%). Within the Negative cohort, flow state was the predominant issue, affecting 54% at Q1 and rising to 76% at Q2.

4.3.2 *Productivity.*

Cross-sectional analysis. At Q1, productivity perceptions were predominantly positive ($M = 4.08$, $SD = 0.65$), with 84% reporting improvement. At Q2, perceptions were similar ($M = 4.03$, $SD = 0.59$), with identical 84% reporting improvement (Figure 8).

Spearman correlations showed a significant moderate positive association between initial impressions and productivity at Q1 ($\rho = 0.31$, $p < 0.001$, moderate effect). At Q2, anticipated disappointment showed a stronger positive correlation ($\rho = 0.48$, $p < 0.001$, moderate effect). Primary concerns showed no significant association with productivity at Q1, but at Q2 productivity perceptions differed by primary concern (Kruskal-Wallis; $\chi^2 = 18.34$, $p = 0.005$, $\varepsilon^2 = 0.19$, large effect), with those citing tool dependency reporting the highest productivity gains.

We next examined the developer experience–productivity relationship through Spearman correlations. All three developer experience dimensions showed significant positive correlations with productivity at both time points (Holm-Bonferroni corrected). At Q1, flow state was strongest ($\rho = 0.49$, $p_{adj} < 0.001$, moderate effect). By Q2, all three

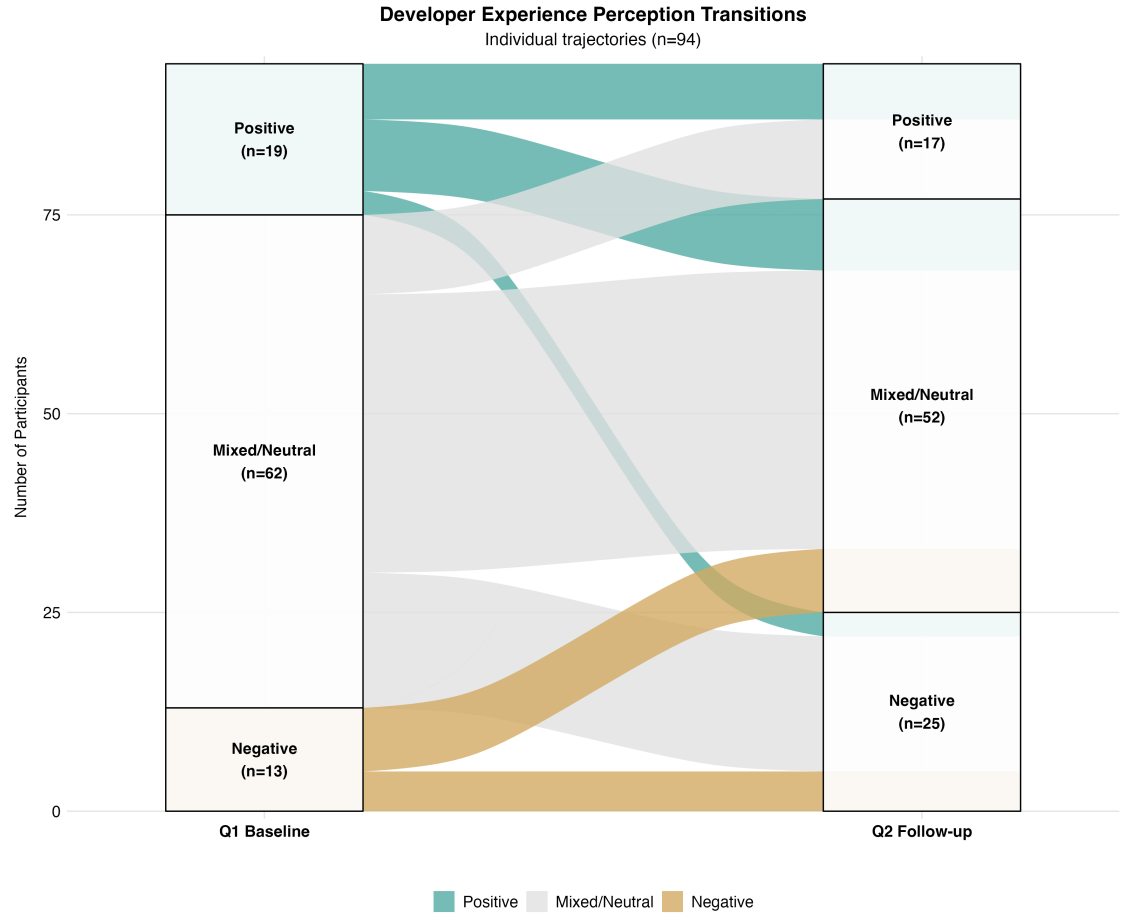


Fig. 7. Developer experience perception transitions between Q1 and Q2.

correlations weakened, and feedback loops replaced flow state as the strongest ($\rho = 0.37$, $p_{adj} < 0.001$, moderate effect), while flow state dropped to a small effect ($\rho = 0.20$, $p_{adj} < 0.05$).

Longitudinal Change in Productivity. Next, we examined how productivity perceptions changed within participants ($n = 95$ matched participants). Paired Wilcoxon signed-rank tests showed no significant change ($\Delta M = -0.05$, $V = 99$, $p = 0.326$, $r_{rb} = -0.22$, small effect). Stability was high: 77% maintained identical ratings, while 14% decreased and 9% increased intensity. No participant transitioned to negative perceptions.

Stability varied by initial position (Figure 9): 85% of "Somewhat better" respondents maintained their rating compared to 65% of "Much better" and 62% of neutral. Movement from neutral was entirely positive, with 38% shifting to "Somewhat better".

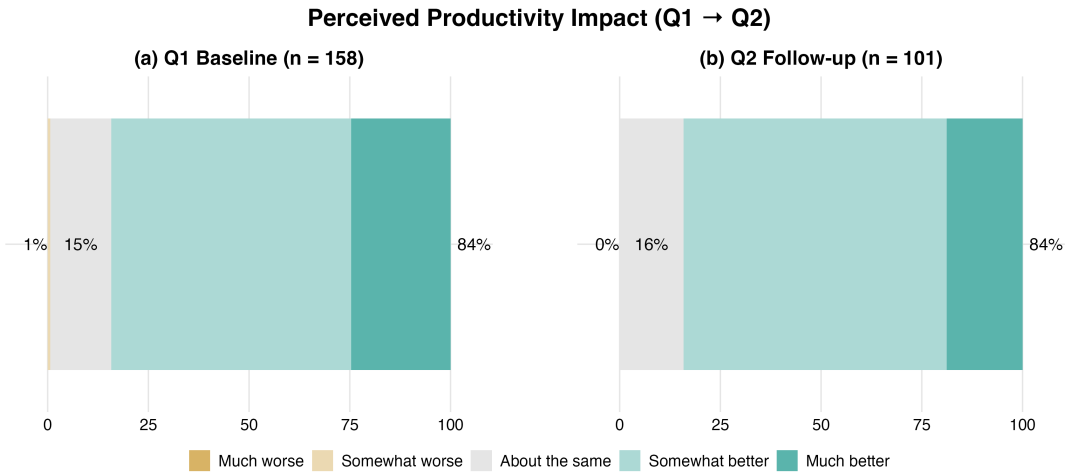


Fig. 8. Perceived productivity impact across time points.

We correlated change scores (Q2 minus Q1) for each developer experience dimension with productivity change ($n = 94$, Holm-Bonferroni corrected). None reached significance (smallest $p_{adj} = 0.313$), contrasting with the moderate-to-strong cross-sectional correlations reported above.

4.3.3 Qualitative Perspectives on Experience and Productivity. Reflexive thematic analysis of responses to the questions asking about instances where AI had significantly helped or hindered work (Q1) or unexpected benefits or challenges (Q2), identified three themes capturing accelerated throughput with uneven experience, expanded capability in unfamiliar work, and verification as ongoing engineering work.

Theme 1: Accelerated throughput with uneven day-to-day experience. Across both time points, participants described AI coding assistants as accelerating progress, particularly for early-stage or repetitive work such as boilerplate generation, test scaffolding, and quick prototyping. Many framed this as a tangible reduction in effort, for example: *"Boring mechanical tasks like refactoring files, updating tests, adding translations, creating test fixtures are 10x less effort with AI assistants"* (P132, Q1). Others described compressed delivery timelines: *"It helped me complete the initial version of the project within hours instead of days . . . It made the entire process much faster and more efficient"* (P91, Q2). AI support was most valued when it enabled rapid starts, reduced mechanical effort, and sustained momentum.

However, participants emphasised that these throughput gains were not uniformly experienced as smoother or easier work. Many described hidden costs in the form of repeated prompting, adaptation effort, or time lost to incorrect output. At Q1, some described prompt loops and rework, for example: *"wasted some time getting into prompt loop"* (P18, Q1). Others reverted to traditional sources when AI was unreliable: *". . . it would reference methods which were no longer relevant, even after specifying a specific version I was using, I ended up going to docs"* (P26, Q1). By Q2, participants continued to describe speed alongside friction. Some described the variability directly: *"Sometimes you gain productivity boost, the other time your wasting time on hallucinations or fixing the generated code"* (P89, Q2). Others noted that flow could be disrupted even as output accelerated, for example: *"This does break the flow, but still speeds up the overall process"* (P28, Q2) and *"less 'in the zone' time, because while code is being produced, I now find myself switching context more often"*

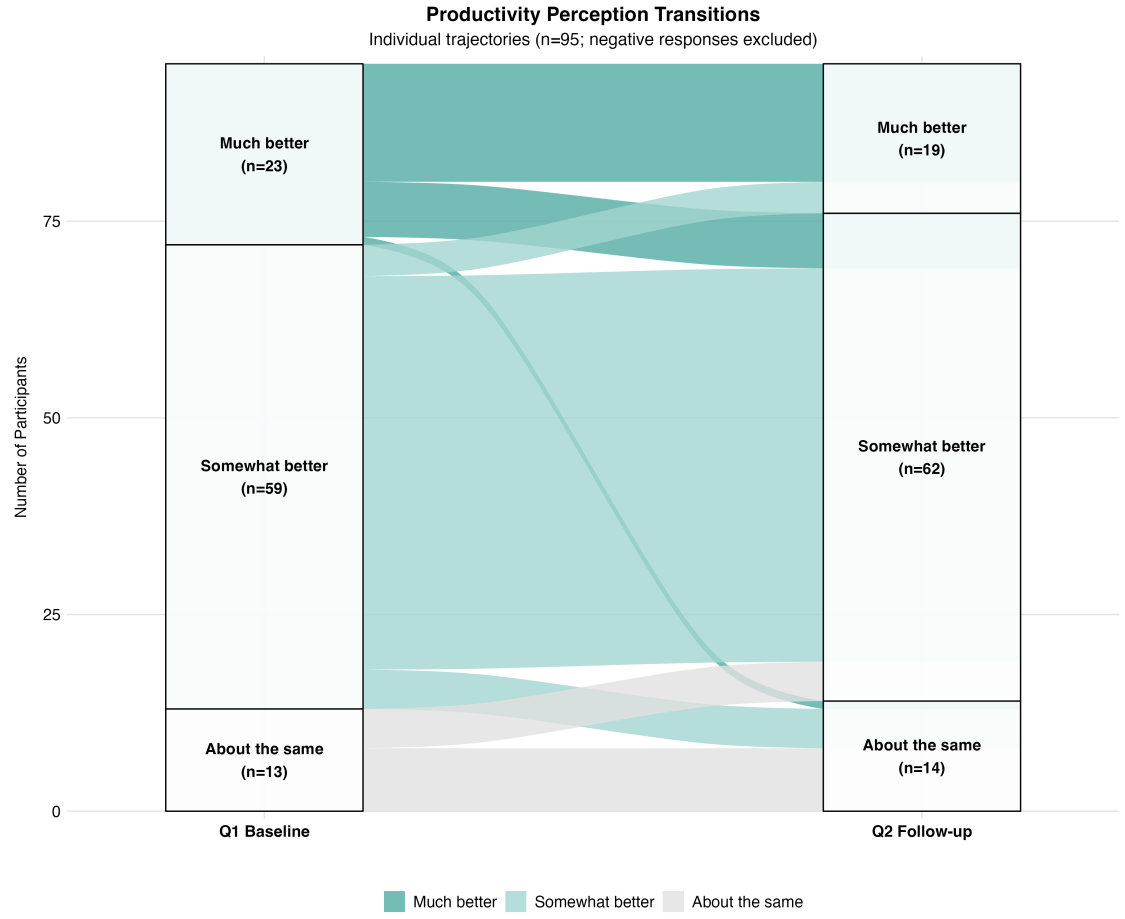


Fig. 9. Productivity perception transitions between Q1 and Q2.

(P159, Q2). Altogether, AI often accelerated throughput while redistributing effort into supervision, correction, and verification. Perceived productivity and lived experience did not always move together.

Theme 2: Expanded perceived capability and willingness to engage with unfamiliar work. Participants described AI coding assistants as lowering the threshold for engaging with unfamiliar technologies, languages, and problem domains at both time points. Rather than claiming mastery, accounts emphasised practical reach, being able to start, explore, or contribute in areas that previously felt inaccessible or too time-consuming to approach. This was often described as a faster path from uncertainty to a workable first step, for example: *"I don't have a lot of experience in authentication flows and got good directions from Gemini on designing a solution"* (P84, Q1) and *"More confident picking up new tools that I'm otherwise unfamiliar with even if they don't have beginner friendly docs"* (P175, Q2).

This expanded capability was framed as reducing hesitation and encouraging experimentation through readily available examples, explanations, and first-pass implementations. However, some cautioned that AI-enabled progress

could short-circuit learning even while increasing output: *"Destroys process of learning something new - you get a working solution, but not understanding why and how it may work"* (P41, Q1). Others cautioned critical judgement and active oversight were needed rather than passive trust: *"I do like the fact that I can ask it about completely unfamiliar domains and it has a reasonable.. sounding at least.. response. I generally feel that I'm critical of all output, but am usually pleasantly surprised"* (P75, Q2). Overall, these accounts position AI coding assistants as expanding perceived capability by making unfamiliar work more approachable and actionable, though sometimes at the expense of deeper understanding. They also offer one reading of why perceived productivity held steady while developer experience ratings declined: the capability gains participants described did not diminish over the study period.

Theme 3: Verification and trust management as ongoing engineering work. AI coding assistants introduced new forms of verification and trust management that remained firmly within the engineer's remit. Participants framed correctness, quality, and consequence as fundamentally human responsibilities rather than treating AI output as authoritative. This required active review, judgement, and intervention. Trust calibration through practices such as sceptical oversight, deliberate steering, and collaborative verification within teams recurred across both time points. These accounts position trust management as an ongoing form of engineering work that shapes how AI-generated code is evaluated, integrated, and governed in practice.

Sub-theme 3A: Individual oversight: scepticism, accountability, and steering practices. Participants framed AI-generated output as provisional, emphasising that responsibility for correctness and consequences remained with the engineer. This accountability manifested through both scepticism and concrete steering practices. At Q1, participants articulated the need for active control, for example: *"You need to always be fully in control, and guide the AI into the direction you want, and not let the AI guide you"* (P101, Q1), with some describing early over-trust as a corrective experience: *"It's gotten code templates completely wrong (yaml) and I was overconfident in the output"* (P129, Q1). This scepticism translated into iterative refinement: *"Often, it takes multiple tries and prompts to achieve the desired results"* (P3, Q1). Steering sometimes required more precise articulation of intent, for example *"E.g. "Track the width of this react component" didn't work, but "Use a react reference to track the width of this react component when it's rendered" did work"* (P82, Q1). By Q2, participants described both recognising model failures and intervening accordingly, for example: *"Identifying when the LLM has lost the thread of the problem and halting it is essential"* (P8, Q2). Constraint-setting became more explicit, such as deliberately breaking work into smaller units to reduce hallucination and drift: *"it's been beneficial to break it down into smaller more specific "modules" and then tie them together at the end"* (P79, Q2). Across both time points, effective AI use was framed as requiring ongoing human judgement—through deliberate scepticism and active steering—rather than increasing reliance or unquestioned trust in AI output.

Sub-theme 3B: Collective trust, reputation, and collaborative implications. Participants described trust management as extending beyond individual practice into shared team contexts, where AI coding assistants affected code review effort, shared understanding, and reputational risk. At Q2, verification concerns extended to reputational and team-level stakes: *"I might use AI chat to help with a question but I do not trust the answer enough for me to hang my reputation on it. I'll always look for a corroborating answer elsewhere"* (P76, Q2). These concerns extended to how participants judged others' AI-generated work, with some expressing reduced confidence in the author's understanding of the code: *"If I know that a dev is using AI heavily I trust them much less to know what exactly their code is doing when reviewing their code"* (P152, Q2), alongside worries about unchecked exploration by less experienced colleagues, such as *"When left unsupervised with AI, junior colleagues sometimes have a tendency to pursue inappropriate solutions to the wrong problems much further"* (P158, Q2). While these concerns were most explicit at Q2, Q1 accounts also signalled the need to retain

human ownership over shared technical direction, for example: *"Hence I keep control of the overall architecture"* (P55, Q1). Altogether, verification was positioned as both an individual and a shared supervisory responsibility, involving additional review discipline, corroboration practices, and clear accountability.

4.3.4 Summary. Productivity perceptions were positive and stable, with 84% reporting improvement at both time points. Developer experience showed cross-sectional associations with productivity: flow state showed the strongest association at Q1 ($\rho = 0.49$), though by Q2 feedback loops became the strongest. Among matched participants, 77% gave identical productivity ratings over six months, while the proportion reporting worsened developer experience in at least one dimension grew from 14% to 27%. Feedback loops improved significantly, while cognitive load and flow state showed non-significant declines. Critically, changes in developer experience did not correlate with changes in productivity. Participants' accounts offer a possible interpretation of this divergence, describing accelerated throughput and expanded capability alongside hidden costs, with trust calibration emerging as ongoing engineering work.

5 Discussion

This section interprets the findings presented in Section 4 and discusses limitations, implications, and directions for future research.

5.1 Task Focus Shifts and Emergence of Supervisory Engineering

Our results indicate that software engineers perceive AI coding assistants as reshaping where they direct their effort across development activities. Writing code showed the clearest decline, with 80% and 82% of participants reporting spending less time on this activity at each time point, respectively. This is consistent with prior research showing AI tools accelerate code generation [15, 42]. Our qualitative findings help to explain where these perceived reductions in time came from. Participants described how AI handles boilerplate, scaffolding, and small fixes that previously consumed substantial effort (Theme 1, *Compressing routine creation work*). They also described turning to AI before Google or Stack Overflow, folding what was once a separate information-gathering step into the coding interaction itself, as well as accelerating onboarding onto unfamiliar codebases, languages, and legacy code (Theme 2, *Rerouting information seeking and comprehension into AI interactions*). Together, these patterns suggest that AI may compress not just the act of writing code but the preparatory work that surrounds it.

Studies using repository activity suggest similar shifts. Analysing open-source projects before and after Copilot's introduction, Xu et al. [61] found that core developers reviewed 6.5% more code while their original code productivity dropped by 19%, measured from logged behaviour rather than self-report. Their setting differs from ours, and they report that the overall productivity increase was driven primarily by less experienced developers, so the convergence lies in the direction of the change rather than in its cause.

Some patterns ran counter to expectations. Given that previous research has found that verification consumes substantial time in AI-assisted development [36], we expected more participants to report spending more time on verification activities like testing and code review. Instead, reviewing code was the only task above neutral at either time point, and only modestly so. Testing remained below neutral throughout, meaning more engineers perceived spending less time on it even in an AI-assisted context. One potential explanation is that engineers do not always recognise the verification they perform on AI-generated output, such as evaluating whether to accept a suggestion, scanning output for errors, or checking that generated code integrates correctly, as "testing" or "code review" in the

traditional sense. As a result, standard Software Development Life Cycle (SDLC) categories may not fully capture where this verification effort is actually going.

Our qualitative findings suggest where some of this effort goes. We propose *supervisory engineering work* (Theme 3, *From doing to supervising AI-generated code*) as an interpretive construct offered for further investigation. It encompasses:

- **Directing**: specifying intent, crafting prompts, providing context, and iterating when output misses the mark
- **Evaluating**: reading AI output and deciding what to accept, modify, or reject
- **Correcting**: fixing errors, integrating output into existing code, and maintaining consistency

Understood this way, supervisory engineering work is broader in kind than the review overhead already reported. That overhead maps onto evaluating and correcting: prior studies describe the effort of assessing AI-generated output [7] and the time taken to review and fix it [4, 56]. Even there the fit is imperfect, since evaluating sits inside authoring rather than after it, with the engineer judging output as it appears in code they are still writing rather than a finished contribution from another author. Directing lies outside it altogether, occurring prior to there being any output to assess and steering what the tool produces rather than judging what it has produced. What we propose is therefore a category that includes review overhead rather than a restatement of it.

Research on the interaction between people and automated systems has examined supervision in other settings. Writing about undersea teleoperators, Sheridan and Verplank [50] introduced the term *supervisory control* for a person directing and monitoring a system that acts on its own between interventions. Parasuraman et al. [41] later extended this concept across the stages of gathering information, analysing it, choosing a course of action, and carrying it out, and argued that automation does not simply remove human work but changes it, creating new coordination demands. This research also documented what the shift cost the people in it. Endsley and Kiris [20] found that operators were slower to take over when an automated decision aid failed, which they attributed to the operator moving from actively working on the problem to passively watching it. They also found that the automation did not reduce the workload participants reported. Bainbridge [2] put it more bluntly, describing a job reduced to monitoring as "very boring but very responsible" and difficult for those in it to come to terms with.

Our construct emerged from our own data in an AI-assisted software engineering setting, which differs in three respects. First, the system being supervised is generative: its output is not measured against a target state the way an industrial process is, so evaluating correctness has to be done on a case by case basis. Second, the supervisory work is itself active rather than the passive monitoring that much of this research treats as the source of these costs. Third, who does what is decided by the engineer, prompt by prompt, within a single task, rather than by a designer or by a scheme that varies with operating conditions [41].

Related work has begun to make this connection: Simkute et al. [51] revisit Bainbridge's ironies for generative AI in knowledge work, and, more recently, Dhanorkar et al. [18] examine oversight practices around autonomous agents. What we propose is more specific: a candidate category of work that emerged from how software engineers described their task focus shifting over the study period.

Since who does what is the engineer's to decide, not all engineers experience this supervisory work equally. Theme 4 from our thematic analysis, *Verification and trust calibration moderating perceived benefits*, captured how engineers vary in their willingness to accept AI output, with some restricting AI to low-risk contexts while others accept suggestions more readily. This calibration shapes how much supervision is performed.

While the quantitative analysis showed a shift in the perceived balance from creation toward verification, the qualitative themes suggested less of a change between time points. The same themes were evident at both, with some

participants placing more or less emphasis on supervision and verification at Q2 (Theme 3, *From doing to supervising AI-generated code*; Theme 4, *Verification and trust calibration moderating perceived benefits*), but no themes appearing or disappearing. One interpretation is that practices evolved gradually over the six months, intensifying existing ways of working rather than replacing them. Furthermore, since the open-ended workflow question at Q2 made no reference to Q1, responses may also have contrasted working with AI against working without it.

More broadly, these findings suggest that AI coding assistants may be reshaping the nature of software development work itself. If engineers spend less time writing code and more time directing and evaluating AI output, this has implications for which skills remain essential, what engineers practice day-to-day, and how the profession develops its next generation.

5.2 Developer Experience, Productivity, and Emergence of Productivity-Experience Paradox

Our results found that software engineers predominantly perceived AI coding assistants as improving their productivity, with 84% reporting improvement at both time points, consistent with industry surveys [52]. Perceived impact on developer experience, using the three DevEx dimensions of cognitive load, feedback loops, and flow state, was also generally positive at Q1, though less uniformly so: 66% reported improved cognitive load, 62% improved feedback loops, and 53% improved flow state. Unlike productivity, developer experience showed concerning signs of erosion over the six-month study period, with the number of matched participants reporting worsened perceptions in at least one dimension increasing from 14% in Q1 to 27%. Our qualitative Theme 1, *Accelerated throughput with uneven day-to-day experience*, helps explain this pattern: participants described impressive time compression alongside persistent friction from repeated prompting and correction.

Flow state was perceived as most vulnerable to erosion. Participants described features of AI-assisted workflows as potentially disruptive to flow. Each AI suggestion requires evaluation; each generation requires verification. The cycle of prompting, reviewing, accepting or rejecting, and iterating can constitute ongoing interruption built into the workflow itself. Our qualitative Theme 3, *Verification and trust management as ongoing engineering work*, captured this dynamic: participants described needing to maintain vigilance when reviewing AI output, a form of mental effort that persists even when the code itself is generated quickly. Prior research found that context switching and interruption frequency are key factors affecting flow [47]. The delegation model may allow developers to turn attention elsewhere while code is being generated, but this shifts work from sustained focus to a rhythm of directing, waiting, and evaluating—precisely the supervisory engineering work we propose is emerging.

Our findings suggest that the established relationships between productivity and developer experience may need re-examination when AI is introduced. We term this pattern the *productivity-experience paradox*: consistently positive productivity perceptions alongside eroding developer experience, potentially decoupling the established relationship between the two. Prior research found that uninterrupted focus is central to developers' sense of productivity [32] and that satisfaction and productivity are bidirectionally linked [54]. Our qualitative findings help explain why productivity held steady while developer experience lowered. Theme 1, *Accelerated throughput with uneven day-to-day experience*, captured participants describing faster output alongside persistent friction, with perceived productivity and lived experience not always moving together. The DevEx framework treats its three dimensions as complementary drivers of productivity, but AI assistance may reshape their relative contributions, with faster feedback compensating for increased cognitive friction. Our correlations show this directly: the correlation between each dimension and perceived productivity weakened between time points, and flow state, the strongest at Q1 ($\rho = 0.49$), was displaced by feedback loops by Q2 while falling to a small effect ($\rho = 0.20$). Changes in the dimensions did not track changes in productivity

at all, with none of the change-score correlations reaching significance. Whether this reconfiguration is sustainable, or whether the erosion in flow and cognitive load eventually undermines the gains, remains an open question.

This paradox describes coexisting perceptions rather than a causal trade-off. The two are parallel self-reports, and our design cannot establish whether one drives the other. The finding matters for what it shows about engineers' lived experience of AI-assisted work, not for any causal mechanism.

5.3 The Shifting Tool Landscape

Our study captured a six-month window in a field evolving at unprecedented speed. The tool landscape shifted substantially even during data collection, with 82% of matched participants changing their tool combinations between time points. Two patterns emerged. First, *tool diversification*: participants mentioned around 37 distinct tools across the study, and engineers are not settling on a single tool but building toolkits. The average number of tools per engineer increased from 1.9 at Q1 to 2.9 at Q2. Second, *shift toward coding-specialised tools*: across our full samples, ChatGPT declined from 70% to 58% usage, while Cursor grew from 16% to 29%. This suggests some engineers are moving from general-purpose conversational AI toward purpose-built coding environments.

The only concern that changed significantly over the six-month study period was maintainability, which increased. This may reflect growing familiarity with AI-generated code, and a recognition that AI optimises for *works now* rather than *maintains later*. One participant echoed this, listing *"is it maintainable"* (P88, Q2) among the checks contributing to their verification time (Theme 4, *Verification and trust calibration moderating perceived benefits*).

5.4 Threats to Validity

Several threats to validity warrant attention when interpreting our findings. First, regarding internal validity, 40% of participants who completed Q1 did not complete Q2, reducing our matched longitudinal sample from 158 to 95. As reported in Section 3, demographic comparisons showed no significant differences between retained and lost participants, supporting the representativeness of the matched cohort. However, we cannot rule out unmeasured differences in attitudes or experiences. Engineers who became disillusioned with AI tools may have been less motivated to complete a follow-up questionnaire about them, which would cause our longitudinal findings to overestimate stability and underestimate negative trajectories. We did not conduct a formal a priori power analysis because we relied on convenience and referral-chain sampling, and the sample size represents the participants who were available during the study period. Non-significant longitudinal results should therefore not be read as evidence of absence: small true effects may have gone undetected.

As disclosed in Section 3, the first author is a practising software engineer whose familiarity with the domain may have influenced the framing of questions, interpretation of findings, or emphasis on certain themes. We managed this threat to internal validity through iterative discussions with the research supervisor throughout data collection and analysis, including survey design and interpretation of both quantitative patterns and qualitative themes. However, this potential bias cannot be completely eliminated.

Threats to external validity include limitations of our sample. Our study only included engineers currently using AI coding assistants; those who tried these tools and abandoned them are absent from our sample. This selection bias likely causes our findings to overestimate positive outcomes, as engineers who found AI unhelpful would have stopped using it and thus would not appear in our data. The 84% positive productivity rate should be interpreted as "84% of continuing users", not "84% of all who tried". Beyond this survivor bias, 85% of our sample were men and predominantly English-speaking, which limits generalisability to under-represented groups. Participation was also geographically

concentrated, with New Zealand accounting for 41% of participants at Q1 and 48% at Q2. The sample is therefore not statistically representative of the global software engineering population, and both self-selection into the study and the platforms we recruited through are likely to have influenced who took part. Finally, some groups were too small to compare separately. For example, only around 2% of participants reported spending more time writing code, too few to characterise as a group, and future work could sample such engineers deliberately.

Another external validity threat is temporal. Our findings reflect experiences with AI coding assistants as they existed in late 2024 and early 2025. Rapid tool evolution means patterns may not hold with newer agentic tools that operate more autonomously. In addition, our data is confounded with a rapidly changing tool landscape; our findings reflect practice-in-the-wild rather than the maturation of any single tool. Changes we observed may stem from tool switching, tool improvement, or growing familiarity, which we cannot fully disentangle; we did not capture how long participants had been using AI coding assistants before the study. Organisational context is a further influence we could not assess, since we did not ask about employer policies or practices around AI use, which may themselves have changed over the study period.

A threat to construct validity is our use of perception-based measures. We captured perceptions rather than objective performance. As discussed in Section 2, perceived and actual productivity can diverge substantially, and several factors may inflate positive perceptions in our data: social desirability, effort justification, and question framing. However, our interpretivist stance treats perceptions as meaningful data in their own right: they drive adoption decisions, shape professional identity, and influence how engineers approach their work.

Single-item measurement poses a related threat to construct validity. With one indicator per construct we cannot estimate measurement error, so small differences between tasks or between time points should be interpreted cautiously.

Participants' understanding of the six development tasks is a further threat to construct validity. As stated in Section 3.1.3, they were presented as labels without definition. Each rating therefore reflects what that task meant to that participant rather than a definition we supplied.

Recall bias is another threat to construct validity. Engineers may overweight memorable events when summarising months of AI use, while recency effects may inflate the influence of experiences closest to each questionnaire. Internal benchmarks may also shift: an engineer who found AI frustrating six months ago may have recalibrated what counts as frustrating, making direct comparisons across time points imprecise. From our interpretivist perspective, however, these distortions are not merely noise to be eliminated but part of what we sought to understand.

Finally, our convergent parallel design limits how far the quantitative and qualitative results can explain one another. Both were collected in the same questionnaire at each time point, so we could not observe a quantitative pattern and then return to participants to probe what lay behind it. Explanatory depth of that kind would require a sequential design, with the qualitative work following the quantitative analysis.

5.5 Implications

5.5.1 For Individual Engineers. The proposed category of supervisory engineering work suggests preparing for a shift in daily practice, with less time writing code and more time directing, evaluating, and correcting AI output. This may be less freed-up time than differently allocated effort. Verification and trust calibration are important to get the most benefits from AI. Knowing when to accept and when to reject AI output emerged across our qualitative findings as an important skill, and one that transfers across the rapidly evolving tool landscape.

5.5.2 For Organisations and Teams. Organisations should recognise that engineers' effort may be shifting rather than simply reducing, from creation toward verification and toward the supervisory engineering work we propose. The perceived effort reallocation has implications for how organisations structure teams, evaluate performance, and prioritise skills when hiring, particularly as verification, judgement, and trust calibration become more central to the role. At the team level, participants described trusting a colleague's understanding of their own code less when they knew AI had done much of the work, and noted junior engineers pursuing unsuitable solutions further when unsupervised. Making AI use visible, adjusting review practices accordingly, and overseeing junior engineers' AI work more deliberately may all help. On measurement, output metrics alone may mask experiential erosion, so monitoring developer experience alongside productivity could provide earlier warning of unsustainable patterns. On tooling, enabling experimentation rather than mandating a single tool may better serve adaptation, given how substantially the landscape shifted even during our six-month study.

5.5.3 For Educators. If AI is shifting engineering work from creation toward verification while introducing new responsibilities like supervisory engineering work, educational programmes may need to prepare students accordingly. This means teaching not just traditional technical skills but also judgement, trust calibration, and the components of supervisory work: directing, evaluating, and correcting AI output. More broadly, transferable principles of human-AI collaboration may prove more durable than tool-specific training given how rapidly the landscape is evolving. Assessment methods may also need to evolve, evaluating understanding rather than output quality alone, since students can now produce working code without necessarily understanding it.

5.5.4 For Tool Builders. As the perceived balance of our participants' work shifted toward verification, so too should the focus of the tools they use. The supervisory engineering work we propose would introduce further requirements, since directing, evaluating and correcting sit outside the traditional development activities and are therefore what today's tools support least well. Interfaces for steering would help with directing, particularly where intent has to be restated before the output is right. In terms of evaluating, engineers would benefit from seeing why a suggestion was produced, as well as being able to tell whether it works and why, since verifying code the engineer did not write is cognitively heavy. Tools could also do more for correcting, helping engineers integrate generated code into the existing solution rather than treating a suggestion as complete once accepted. The agentic tools that have emerged since our study would place even further demands on tools, since an engineer cannot direct, evaluate, or correct what they cannot see: understanding what such a tool is doing, how far it has progressed, and when it has drifted from the problem matter as much as the quality of any individual suggestion. On developer experience, only feedback loops improved significantly, so fast iteration appears well served; cognitive load and flow state did not. Flow state was the predominant issue in the Negative cohort, at 76% by Q2, which suggests a need to focus on protecting concentration and reducing interruption as core features of the tools engineers use when they build with AI.

5.6 Future Research Directions

5.6.1 Emerging Competencies and Work Categories. AI assistance may be introducing new work categories that existing frameworks do not capture. The supervisory engineering work we proposed raises questions about whether this represents a genuinely new SDLC activity distinct from traditional categories like designing, coding, testing, and reviewing. Future research could examine whether supervisory engineering work is empirically distinct from existing activities, what skills it requires, and how engineers should develop them. If this work category proves real and distinct, established SDLC models may need updating to account for where developer effort now goes. That effort may also

be distributed differently between engineers. Recent research has found that students with different cognitive styles engage with AI coding assistants in different ways, including differences in validation behaviour, trust formation, and willingness to rely on AI-generated solutions [26]. Future research could examine whether professional engineers similarly adopt distinct work styles when supervising AI, for example placing more emphasis on directing, evaluating or correcting, and what factors shape those styles.

5.6.2 Developer Experience and Productivity. Prior research has shown that developer experience drives productivity, yet our data suggests a productivity-experience paradox, with productivity holding steady while two of the three DevEx dimensions worsened. Future research could examine whether the DevEx dimensions contribute differently in AI-assisted contexts, whether new dimensions are needed, and whether the relationship between developer experience and productivity has changed. Longer-term studies tracking both productivity and experience could illuminate whether the divergence we observed is sustainable over longer time horizons or whether worsening flow state and cognitive load manifest as burnout or turnover.

5.6.3 Satisfaction in Supervisory Work. The shift toward supervisory engineering work raises questions about developer satisfaction. The worsening flow state and cognitive load we observed suggest some experiential costs may accompany this transition. Future research could examine whether engineers find supervisory work intrinsically satisfying or whether reduced hands-on coding diminishes professional fulfilment and whether these effects vary across developers. If supervisory work proves less satisfying, interventions could be devised to improve satisfaction.

5.6.4 Team and Organisational Dynamics. Our study focused on individual engineers, but the creation-to-verification shift and supervisory engineering work we described likely have team-level implications. Research examining team dynamics and organisational structures could illuminate how individual-level findings aggregate at the team level and whether supervisory engineering and verification efforts are equally distributed.

6 Conclusion

This study examined how professional software engineers perceive the effects of AI coding assistants on their work, tracking those perceptions over time. Using a longitudinal mixed-methods design with questionnaires administered at two time points six months apart allowed us to see shifts that cross-sectional snapshots would miss.

Our findings suggest that development work is being reorganised rather than simply accelerated. Most engineers perceived spending less time across traditional development tasks, with the relative balance shifting from creation toward verification; we have termed this the *creation-to-verification shift*. Yet the increase in verification was modest, raising the question of where the effort is going. We propose this effort flows into what we have called *supervisory engineering work*: directing AI, evaluating its output, and correcting its errors. Meanwhile, the *productivity-experience paradox* we documented suggests these tools may be decoupling dimensions that previously moved together, with productivity perceptions remaining consistently positive even as developer experience showed signs of erosion over time, particularly in flow state and cognitive load. Together, these patterns suggest that AI coding assistants may be reshaping the nature of software engineering work, not merely making existing tasks faster.

Whether AI coding assistants will finally close the persistent gap between ambition and achievement that has defined software engineering since its origins remains to be seen. Brooks warned that technology alone has historically failed to deliver dramatic improvements [10], and our findings suggest that unintended consequences may be emerging. As software engineering serves as a proving ground for AI-assisted work, these patterns may foreshadow dynamics that

other professions will encounter as AI tools spread. For a global community of professionals who have built careers around the craft of building software, understanding how engineers perceive these tools is essential, because perceptions drive behaviour. This study contributes to that understanding, informing educators preparing the next generation, organisations supporting their teams, those building the tools themselves, and engineers navigating their own paths through this period of rapid change.

Replication Package

The replication package for this study is openly available [58]. It contains the full questionnaire instrument administered at both time points, the codebook developed during the qualitative analysis, the R packages used for the quantitative analysis, and the full list of AI coding assistants participants reported using. Readers can use this to see exactly what was asked of participants, how their open-ended responses were coded, and which tools lie behind the figures reported in Section 4. Individual responses are not included, because participants did not consent to their responses being shared.

Use of GenAI

We used Claude Code (Anthropic) to assist with developing R scripts for statistical analysis and reviewing written text for clarity and readability. All outputs were critically reviewed and validated, and responsibility for the correctness of the analysis and interpretations rests entirely with the researchers.

Acknowledgments

We thank all participants for their time and input into this research. This research was partially supported by Westpac New Zealand, Annie Vella's employer at the time the research was conducted, and a Rutherford Discovery Fellowship administered by the Royal Society Te Apārangi, New Zealand.

References

[1] Adam Alami and Neil Ernst. 2025. Human and Machine: How Software Engineers Perceive and Engage with AI-Assisted Code Reviews Compared to Their Peers. In *2025 IEEE/ACM 18th International Conference on Cooperative and Human Aspects of Software Engineering (CHASE)*. IEEE, New York, NY, USA, 63–74. doi:10.1109/CHASE66643.2025.00016

[2] Lisanne Bainbridge. 1983. Ironies of automation. *Automatica* 19, 6 (Nov. 1983), 775–779. doi:10.1016/0005-1098(83)90046-8

[3] Sebastian Baltes and Paul Ralph. 2022. Sampling in software engineering research: a critical review and guidelines. *Empirical Software Engineering* 27, 94 (April 2022), 1–31. doi:10.1007/s10664-021-10072-8

[4] Leonardo Banh, Florian Holldack, and Gero Strobel. 2025. Copiloting the future: How generative AI transforms Software Engineering. *Information and Software Technology* 183 (July 2025), 107751. doi:10.1016/j.infsof.2025.107751

[5] Shraddha Barke, Michael B. James, and Nadia Polikarpova. 2023. Grounded Copilot: How Programmers Interact with Code-Generating Models. *Proceedings of the ACM on Programming Languages* 7, OOPSLA1 (April 2023), 85–111. doi:10.1145/3586030

[6] Joel Becker, Nate Rush, Beth Barnes, and David Rein. 2025. Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity. METR. https://metr.org/blog/2025-07-10-early-2025-ai-experienced-os-dev-study/

[7] Christian Bird, Denae Ford, Thomas Zimmermann, Nicole Forsgren, Eirini Kalliamvakou, Travis Lowdermilk, and Idan Gazit. 2022. Taking Flight with Copilot: Early insights and opportunities of AI-powered pair-programming tools. *Queue* 20, 6 (Dec. 2022), 35–57. doi:10.1145/3582083

[8] Virginia Braun and Victoria Clarke. 2006. Using thematic analysis in psychology. *Qualitative Research in Psychology* 3, 2 (Jan. 2006), 77–101. doi:10.1191/1478088706qp063oa

[9] Virginia Braun and Victoria Clarke. 2019. Reflecting on reflexive thematic analysis. *Qualitative Research in Sport, Exercise and Health* 11, 4 (Aug. 2019), 589–597. doi:10.1080/2159676x.2019.1628806

[10] Brooks. 1987. No Silver Bullet Essence and Accidents of Software Engineering. *Computer* 20, 4 (April 1987), 10–19. doi:10.1109/mc.1987.1663532

[11] Jenna Butler, Jina Suh, Sankeerti Haniyur, and Constance Hadley. 2025. Dear Diary: A Randomized Controlled Trial of Generative AI Coding Tools in the Workplace. In *2025 IEEE/ACM 47th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, New York, NY, USA, 319–329. doi:10.1109/ICSE-SEIP66354.2025.00034

[12] Sayan Chatterjee, Ching Louis Liu, Gareth Rowland, and Tim Hogarth. 2024. The Impact of AI Tool on Engineering at ANZ Bank: An Empirical Study on GitHub Copilot within Corporate Environment. doi:10.48550/arXiv.2402.05636

[13] Tianyi Chen. 2024. The Impact of AI-Pair Programmers on Code Quality and Developer Satisfaction: Evidence from TiMi studio. In *2024 International Conference on Generative Artificial Intelligence and Information Security (GAIIS)*. ACM, New York, NY, USA, 201–205. doi:10.1145/3665348.3665383

[14] John W. Creswell and Vicki L. Plano Clark. 2018. *Designing and Conducting Mixed Methods Research* (third ed.). SAGE, Thousand Oaks, California, USA.

[15] Zheyuan (Kevin) Cui, Mert Demirer, Sonia Jaffe, Leon Musolff, Sida Peng, and Tobias Salz. 2025. The Effects of Generative AI on High-Skilled Work: Evidence from Three Field Experiments with Software Developers. doi:10.2139/ssrn.4945566

[16] Sarah D'Angelo, Ambar Murillo, Satish Chandra, and Andrew Macvean. 2024. What Do Developers Want From AI? *IEEE Software* 41, 3 (May 2024), 11–15. doi:10.1109/MS.2024.3363538

[17] Kayla DePalma, Izabel Miminoshvili, Chiara Henselder, Kate Moss, and Eman Abdullah AlOmar. 2024. Exploring ChatGPT's code refactoring capabilities: An empirical study. *Expert Systems with Applications* 249 (Sept. 2024), 123602. doi:10.1016/j.eswa.2024.123602

[18] Shipi Dhanorkar, Samir Passi, and Mihaela Vorvoreanu. 2026. Human oversight of agentic systems in practice: Examining the oversight work, challenges, and heuristics of developers using software agents. doi:10.48550/ARXIV.2606.05391

[19] DORA. 2025. *State of AI-assisted Software Development.* Technical Report. Google. https://dora.dev/research/2025/dora-report/

[20] Mica R. Endsley and Esin O. Kiris. 1995. The Out-of-the-Loop Performance Problem and Level of Control in Automation. *Human Factors: The Journal of the Human Factors and Ergonomics Society* 37, 2 (June 1995), 381–394. doi:10.1518/001872095779064555

[21] Manuel Hoffmann, Sam Boysel, Frank Nagle, Sida Peng, and Kevin Xu. 2025. Generative AI and the Nature of Work. doi:10.2139/ssrn.5007084

[22] Brian Houck, Travis Lowdermilk, Cody Beyer, Steven Clarke, and Ben Hanrahan. 2025. The SPACE of AI: Real-World Lessons on AI's Impact on Developers. doi:10.48550/arXiv.2508.00178

[23] Anna Y. Q. Huang, Cheng-Yan Lin, Sheng-Yi Su, and Stephen J. H. Yang. 2025. The impact of GenAI-enabled coding hints on students' programming performance and cognitive load in an SRL-based Python course. *British Journal of Educational Technology* 56, 5 (2025), 1942–1972. doi:10.1111/bjet.13589

[24] Sarah Inman, Ambar Murillo, Sarah D'Angelo, Adam Brown, and Collin Green. 2025. Seamful AI for Creative Software Engineering. *IEEE Software* 42 (2025), 59–66. doi:10.1109/MS.2025.3534085

[25] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2023. SWE-bench: Can Language Models Resolve Real-World GitHub Issues? doi:10.48550/arXiv.2310.06770

[26] Grace Kerr, Aarushi Parasrampuria, Faith Culas, and Kelly Blincoe. 2026. Cognitive Inclusivity in AI Coding Assistants: Evidence from GitHub Copilot. arXiv:2610.41192 [cs.SE] https://arxiv.org/abs/2610.41192

[27] Mohammad Amin Kuhail, Sujith Samuel Mathew, Ashraf Khalil, Jose Berengueres, and Syed Jawad Hussain Shah. 2024. "Will I be replaced?" Assessing ChatGPT's effect on software development and programmer perceptions of AI tools. *Science of Computer Programming* 235 (July 2024), 103111. doi:10.1016/j.scico.2024.103111

[28] Anand Kumar, Vishal Khare, Deepak Sharma, Satyam Kumar, Vijay Saini, Anshul Yadav, Sachendra Jain, Ankit Rana, Pratham Verma, Vaibhav Meena, and Avinash Edubilli. 2025. Intuition to Evidence: Measuring AI's True Impact on Developer Productivity. doi:10.48550/arXiv.2509.19708

[29] Eve Martina Lange, Åsa Cajander, and Maria Normark. 2025. Exploring Flow in IT Professionals' Use of AI-Integrated Tools: Insights from Interviews. In *Artificial Intelligence in HCI*, Helmut Degen and Stavroula Ntoa (Eds.). Springer Nature Switzerland, Cham, 44–58. doi:10.1007/978-3-031-93429-2_3

[30] Jenny T. Liang, Chenyang Yang, and Brad A. Myers. 2024. A Large-Scale Survey on the Usability of AI Programming Assistants: Successes and Challenges. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering (ICSE '24)*. Association for Computing Machinery, New York, NY, USA, 1–13. doi:10.1145/3597503.3608128

[31] André N. Meyer, Laura E. Barton, Gail C. Murphy, Thomas Zimmermann, and Thomas Fritz. 2017. The Work Life of Developers: Activities, Switches, and Perceived Productivity. *IEEE Transactions on Software Engineering* 43, 12 (2017), 1178–1193. doi:10.1109/TSE.2017.2656886

[32] André N. Meyer, Thomas Fritz, Gail C. Murphy, and Thomas Zimmermann. 2014. Software developers' perceptions of productivity. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2014)*. Association for Computing Machinery, New York, NY, USA, 19–29. doi:10.1145/2635868.2635892

[33] Amr Mohamed, Maram Assi, and Mariam Guizani. 2026. The Impact of LLM-Assistants on Software Developer Productivity: A Systematic Review and Mapping Study. *ACM Transactions on Software Engineering and Methodology* (April 2026). doi:10.1145/3809494

[34] Hal Mooz and Kevin Forsberg. 2006. 10.2.1 The Dual Vee - Illuminating the Management of Complexity. *INCOSE International Symposium* 16, 1 (2006), 1368–1381. doi:10.1002/j.2334-5837.2006.tb02819.x

[35] Arghavan Moradi Dakhel, Vahid Majdinasab, Amin Nikanjam, Foutse Khomh, Michel C. Desmarais, and Zhen Ming (Jack) Jiang. 2023. GitHub Copilot AI pair programmer: Asset or Liability? *Journal of Systems and Software* 203 (Sept. 2023), 111734. doi:10.1016/j.jss.2023.111734

[36] Hussein Mozannar, Gagan Bansal, Adam Fourney, and Eric Horvitz. 2024. Reading Between the Lines: Modeling User Behavior and Costs in AI-Assisted Programming. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems (CHI '24)*. Association for Computing Machinery, New York, NY, USA, 1–16. doi:10.1145/3613904.3641936

[37] Emerson Murphy-Hill, Ciera Jaspan, Caitlin Sadowski, David Shepherd, Michael Phillips, Collin Winter, Andrea Knight, Edward Smith, and Matthew Jorde. 2021. What Predicts Software Developers' Productivity? *IEEE Transactions on Software Engineering* 47, 3 (March 2021), 582–594. doi:10.1109/TSE.2019.2900308

[38] Nhan Nguyen and Sarah Nadi. 2022. An empirical evaluation of GitHub copilot's code suggestions. In *Proceedings of the 19th International Conference on Mining Software Repositories*. ACM, Pittsburgh, PA, USA, 1–5. doi:10.1145/3524842.3528470

[39] Abi Noda, Margaret-Anne Storey, Nicole Forsgren, and Michaela Greiler. 2023. DevEx: What Actually Drives Productivity: The developer-centric approach to measuring and improving productivity. *Queue* 21, 2 (May 2023), Pages 20:35–Pages 20:53. doi:10.1145/3595878

[40] Ruchika Pandey, Prabhat Singh, Raymond Wei, and Shaila Shankar. 2024. Transforming Software Development: Evaluating the Efficiency and Challenges of GitHub Copilot in Real-World Projects. arXiv:2406.17910 [cs.SE] https://arxiv.org/abs/2406.17910

[41] R. Parasuraman, T.B. Sheridan, and C.D. Wickens. 2000. A model for types and levels of human interaction with automation. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans* 30, 3 (May 2000), 286–297. doi:10.1109/3468.844354

[42] Sida Peng, Eirini Kalliamvakou, Peter Cihon, and Mert Demirer. 2023. The Impact of AI on Developer Productivity: Evidence from GitHub Copilot. arXiv:2302.06590 [cs.SE] https://arxiv.org/abs/2302.06590

[43] Neil Perry, Megha Srivastava, Deepak Kumar, and Dan Boneh. 2023. Do Users Write More Insecure Code with AI Assistants?. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS '23)*. Association for Computing Machinery, New York, NY, USA, 2785–2799. doi:10.1145/3576915.3623157

[44] Gustavo Pinto, Cleidson De Souza, Thayssa Rocha, Igor Steinmacher, Alberto Souza, and Edward Monteiro. 2024. Developer Experiences with a Contextualized AI Coding Assistant: Usability, Expectations, and Outcomes. In *Proceedings of the IEEE/ACM 3rd International Conference on AI Engineering - Software Engineering for AI (CAIN '24)*. Association for Computing Machinery, New York, NY, USA, 81–91. doi:10.1145/3644815.3644949

[45] James Prather, Brent N. Reeves, Juho Leinonen, Stephen MacNeil, Arisoa S. Randrianasolo, Brett A. Becker, Bailey Kimmel, Jared Wright, and Ben Briggs. 2024. The Widening Gap: The Benefits and Harms of Generative AI for Novice Programmers. In *Proceedings of the 2024 ACM Conference on International Computing Education Research*, Vol. 1. ACM, New York, NY, USA, 469–486. doi:10.1145/3632620.3671116

[46] Sanka Rasnayaka, Guanlin Wang, Ridwan Shariffdeen, and Ganesh Neelakanta Iyer. 2024. An Empirical Study on Usage and Perceptions of LLMs in a Software Engineering Project. doi:10.48550/arXiv.2401.16186

[47] Abdul Razzaq, Jim Buckley, Qin Lai, Tingting Yu, and Goetz Botterweck. 2024. A Systematic Literature Review on the Influence of Enhanced Developer Experience on Developers' Productivity: Factors, Practices, and Recommendations. *ACM Comput. Surv.* 57, 1 (Oct. 2024), 13:1–13:46. doi:10.1145/3687299

[48] Agnia Sergeyuk, Yaroslav Golubev, Timofey Bryksin, and Iftekhar Ahmed. 2025. Using AI-based coding assistants in practice: State of affairs, perceptions, and ways forward. *Information and Software Technology* 178 (Feb. 2025), 107610. doi:10.1016/j.infsof.2024.107610

[49] Agnia Sergeyuk, Eric Huang, Dariia Karaeva, Anastasiia Serova, Yaroslav Golubev, and Iftekhar Ahmed. 2026. Evolving with AI: A Longitudinal Analysis of Developer Logs. doi:10.1145/3744916.3787811

[50] Thomas B. Sheridan and William L. Verplank. 1978. *Human and Computer Control of Undersea Teleoperators:*. Technical Report. Defense Technical Information Center, Fort Belvoir, VA, USA. doi:10.21236/ADA057655

[51] Auste Simkute, Lev Tankelevitch, Viktor Kewenig, Ava Elizabeth Scott, Abigail Sellen, and Sean Rintel. 2025. Ironies of Generative AI: Understanding and Mitigating Productivity Loss in Human-AI Interaction. *International Journal of Human-Computer Interaction* 41, 5 (2025), 2898–2919. doi:10.1080/10447318.2024.2405782

[52] Stack Overflow. 2024. AI | 2024 Stack Overflow Developer Survey. https://survey.stackoverflow.co/2024/ai

[53] Stack Overflow. 2025. 2025 Stack Overflow Developer Survey. https://survey.stackoverflow.co/2025/

[54] Margaret-Anne Storey, Thomas Zimmermann, Christian Bird, Jacek Czerwonka, Brendan Murphy, and Eirini Kalliamvakou. 2021. Towards a Theory of Software Developer Job Satisfaction and Perceived Productivity. *IEEE Transactions on Software Engineering* 47, 10 (Oct. 2021), 2125–2142. doi:10.1109/TSE.2019.2944354

[55] Viktoria Stray, Elias Goldmann Brandtzæg, Viggo Wivestad, Astri Barbala, and Nils Brede Moe. 2026. Developer Productivity With and Without GitHub Copilot: A Longitudinal Mixed-Methods Case Study. In *Proceedings of the 59th Hawaii International Conference on System Sciences (HICSS)*. 7413–7422. doi:10.24251/HICSS.2026.880

[56] Thiago S. Vaillant, Felipe Deveza de Almeida, Paulo Anselmo M. S. Neto, Cuiyun Gao, Jan Bosch, and Eduardo Santana de Almeida. 2024. Developers' Perceptions on the Impact of ChatGPT in Software Development: A Survey. http://arxiv.org/abs/2405.12195

[57] Priyan Vaithilingam, Tianyi Zhang, and Elena L. Glassman. 2022. Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models. In *CHI Conference on Human Factors in Computing Systems Extended Abstracts*. ACM, New Orleans, LA, USA, 1–7. doi:10.1145/3491101.3519665

[58] Annie Vella and Kelly Blincoe. 2026. Replication package for "The Impact of AI Coding Assistants on Software Engineering: A Longitudinal Study". https://doi.org/10.5281/zenodo.21896822

[59] Thomas Weber, Maximilian Brandmaier, Albrecht Schmidt, and Sven Mayer. 2024. Significant Productivity Gains through Programming with Large Language Models. *Proc. ACM Hum.-Comput. Interact.* 8, EICS (June 2024), 256:1–256:29. doi:10.1145/3661145

[60] Justin D. Weisz, Shraddha Kumar, Michael Muller, Karen-Ellen Browne, Arielle Goldberg, Ellice Heintze, and Shagun Bajpai. 2024. Examining the Use and Impact of an AI Code Assistant on Developer Productivity and Experience in the Enterprise. doi:10.48550/arXiv.2412.06603

[61] Feiyang Xu, Poonacha K. Medappa, Murat M. Tunc, Martijn Vroegindeweij, and Jan C. Fransoo. 2026. AI-Assisted Programming Decreases the Productivity of Experienced Developers by Increasing the Technical Debt and Maintenance Burden. doi:10.48550/arXiv.2510.10165

[62] Burak Yetiştiren, Işık Özsoy, Miray Ayerdem, and Eray Tüzün. 2023. Evaluating the Code Quality of AI-Assisted Code Generation Tools: An Empirical Study on GitHub Copilot, Amazon CodeWhisperer, and ChatGPT. arXiv:2304.10778 [cs.SE] https://arxiv.org/abs/2304.10778

[63] Han Zhang, Shiyi Wang, and Zijian Li. 2025. The Neurophysiological Paradox of AI-Induced Frustration: A Multimodal Study of Heart Rate Variability, Affective Responses, and Creative Output. *Brain Sciences* 15, 6 (May 2025), 565. doi:10.3390/brainsci15060565

[64] Albert Ziegler, Eirini Kalliamvakou, X. Alice Li, Andrew Rice, Devon Rifkin, Shawn Simister, Ganesh Sittampalam, and Edward Aftandilian. 2022. Productivity assessment of neural code completion. In *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*. ACM, New York, NY, USA, 21–29. doi:10.1145/3520312.3534864