Journal of Information Systems Education

Volume 37
Issue 4
Fall 2026

From Line-By-Line to Vibe Coding: Teaching Orchestration in Information Systems Education

Jeremiah Bohr

Recommended Citation: Bohr, J. (2026). From Line-By-Line to Vibe Coding: Teaching Orchestration in Information Systems Education. *Journal of Information Systems Education*, 37(4), 507-525. https://doi.org/10.62273/GFKF5366

Article Link: https://jise.org/Volume37/n4/JISE2026v37n4pp507-525.html

Received:	October 20, 2025
First Decision:	November 27, 2025
Accepted:	January 14, 2026
Published:	December 15, 2026

Find archived papers, submission instructions, terms of use, and much more at the JISE website:
https://jise.org

ISSN: 2574-3872 (Online) 1055-3096 (Print)

Journal of Information Systems Education, 37(4), 507-525, Fall 2026
https://doi.org/10.62273/GFKF5366

From Line-By-Line to Vibe Coding: Teaching Orchestration in Information Systems Education

Jeremiah Bohr
School of Business
University of Wisconsin - Oshkosh
Oshkosh, WI 54901, USA
bohrj@uwosh.edu

ABSTRACT

Software development is shifting from line-by-line authorship to intent-mediated workflows ("vibe coding") that require developers to orchestrate, evaluate, and integrate model outputs. This paper argues that information systems curricula should pivot from producing more code to reasoning more effectively in AI-assisted coding. This theory-informed descriptive study synthesizes trust-in-automation research with active-learning frameworks to reframe AI-assisted coding as a calibration problem of aligning reliance with demonstrated system capability, rather than a generation problem. Descriptive evidence from the 2023–2025 Stack Overflow Developer Survey provides motivational context: adoption is rising while stated trust declines (a widening usage-trust gap); trust differences across experience levels narrow by 2025; frustrations concentrate in accuracy, explanation, and integration; learning occurs primarily through informal channels; and vibe coding is broadly shared across experience levels. Integrating these patterns with theory, four low-overhead activity types are proposed for pilot testing in programming-intensive IS courses: (1) calibration sprints that move "almost-right" outputs to verified solutions, (2) spec-to-code exercises with explicit constraints and rationales, (3) integration diaries with lightweight automated checks, and (4) structured code reviews with roles and explicit accept/revise/reject decisions. The goal is to equip students to use fluent but fallible systems with calibrated reliance and reproducible process, keeping coursework aligned with evolving development practice.

Keywords: Natural language programming, Generative AI, Information systems education, Human-computer interaction (HCI)

1. INTRODUCTION

Software development is shifting from line-by-line production to intent-mediated collaboration with generative AI systems. This mode of working, recently coined "vibe coding," involves developers stating goals and constraints in plain language and guiding an AI to produce code, while remaining responsible for what ships (Meske et al., 2025). The locus of expertise moves from writing every line to orchestrating, evaluating, and integrating model contributions: writing tests, reviewing changes, and ensuring architectural and security fit (Gadde, 2025; Meske et al., 2025).

Adoption of AI-assisted coding is rising rapidly, yet performance impacts vary by context and experience. Randomized trials inside firms report gains in task completion for software developers, often largest for lower-tenure engineers. However, a randomized field experiment with experienced open-source contributors working in mature repositories finds slower completion when assistants are allowed, plausibly due to verification load and almost-right outputs (Becker et al., 2025; Cui et al., 2025). Related evidence

shows that subjective confidence and objective outcomes can diverge in human-AI collaboration, reinforcing the verification-load interpretation (Qian & Wexler, 2024). U.S. national survey waves show rapid workplace uptake from late 2024 to mid-2025, and a large Danish survey finds pervasive adoption in exposed occupations, underscoring the stakes for novice preparation (Hartley et al., 2025; Humlum & Vestergaard, 2024). Together, these findings call for educational approaches that prioritize verification, uncertainty monitoring, and integration over prompting alone.

Viewed through an Information Systems lens, AI-assisted coding is embedded in educational and organizational contexts rather than representing a simple tooling change (Rose et al., 2023). Parallel discussions in higher education emphasize pedagogical and ethical guidance for generative AI adoption (Gimpel et al., 2025), yet systematic evidence on how developers actually learn and calibrate these practices remains scarce. This shift raises curricular questions relevant to programming-intensive IS courses: Where do developers currently learn these practices? How do patterns of usage and trust characterize current adoption? And how prevalent are intent-to-implementation workflows among developers today?

This paper is a theory-informed descriptive study with pedagogical implications. It uses data from the 2023–2025 Stack Overflow Developer Survey (SODS) to characterize current patterns in AI-assisted coding adoption, trust, frustrations, and learning channels (The URLs for the three years of data are as follows: https://survey.stackoverflow.co/2025/; https://survey.stackoverflow.co/2024/; https://survey.stackoverflow.co/2023/). These descriptive findings provide motivational context for the paper's primary contribution: a synthesis of trust-in-automation research and active-learning frameworks that yields concrete, low-overhead classroom activities designed to cultivate calibrated reliance aligning trust with demonstrated system capability, rather than unstructured code generation. The proposed activities are designed for pilot testing in programming-intensive IS courses; they have not yet been empirically evaluated.

This study addresses three questions:

1) Where do developers report learning AI-assisted coding?
2) What patterns in usage, trust, and experience characterize current AI coding tool adoption?
3) How prevalent is vibe coding (intent-to-implementation workflows) among developers in 2025?

This study makes three contributions. First, it synthesizes trust-in-automation theory (Lee & See, 2004; Parasuraman & Riley, 1997) with active-learning frameworks (Chi & Wylie, 2014; Crouch & Mazur, 2001) to reframe AI-assisted coding as a calibration problem rather than a generation problem, with direct implications for course sequencing and assessment. Second, it translates this theoretical synthesis into four ready-to-pilot activity types: calibration sprints that move "almost-right" outputs to verified solutions, spec-to-code exercises with explicit constraints and rationales, integration diaries with lightweight automated checks, and structured code reviews with roles and explicit accept/revise/reject decisions. Third, it provides descriptive evidence from SODS (2023–2025) on adoption trends, trust patterns, reported frustrations, and learning channels that situates the pedagogical proposals within current industry practice and highlights an opportunity for formal education to provide structured evaluation routines that informal learning channels typically lack.

The remainder proceeds as follows. Section 2 situates the work within research on AI-assisted coding, trust in automation, and active learning. Section 3 describes data and measures. Section 4 reports temporal patterns in adoption and trust, as well as reported frustrations, learning channels, and vibe coding prevalence. Section 5 interprets these findings, and Section 6 outlines pedagogical implications.

2. LITERATURE REVIEW

2.1 AI-Assisted Coding and Intent-Mediated Workflows

In AI-assisted coding, expertise shifts from writing every line to coordinating and judging AI output: writing tests that define "correct," reviewing changes, and checking that generated code fits architectural and security requirements (Meske et al., 2025). A central tension is that models generate code by probabilistic inference, yet programs execute deterministically; managing that uncertainty demands disciplined

specification, testing, and review (Meske et al., 2025; Ray, 2025). Recent practitioner syntheses based on industry interviews indicate that, in practice, these workflows often prioritize speed and accessibility while skipping basic quality assurance steps, heightening the need for evaluation practices (Fawzy et al., 2025). Governing AI-assisted coding calls for organizational controls and evaluation practices that address bias, transparency/accountability, and dependency risks—supported by clear attribution mechanisms, standardized evaluation metrics, and technical review processes for generated code (Gadde, 2025). From an Information Systems vantage point, Rose et al. (2023) found that faculty viewed generative tools like ChatGPT with ambivalence: they acknowledged potential benefits for productivity and coding support, but raised strong concerns about plagiarism, code quality, and the devaluation of degrees, underscoring that AI-assisted coding alters educational integrity and professional preparation rather than being a simple tooling upgrade.

It is useful to distinguish AI-assisted coding from more autonomous "agentic" automation. Agentic systems pursue goals across tools and steps with far less oversight; AI-assisted workflows keep a human in the loop to break work into smaller tasks and evaluate each result (Sapkota et al., 2025). Practice accounts report behaviors such as re-prompting instead of debugging and accepting AI outputs without verification, underscoring the need to cast the human role as explicitly evaluative (Fawzy et al., 2025). This distinction suggests curriculum sequencing: begin with conversational, human-in-the-loop orchestration and evaluation, and then add agentic patterns once students can state what "done" looks like and interpret test results, such as automating large-scale refactoring and running checks before release (Ray, 2025; Sapkota et al., 2025). Across these accounts, the educational implication is consistent: if generative tools handle much of the first draft, programs should emphasize higher-order practices—test design, review of code changes, integration work, and policy checks—that keep code reliable, secure, and maintainable (Gadde, 2025; Meske et al., 2025). The literature converges on a pivot in professional competence: from typing code to articulating intent, constraining the search space, and judging adequacy—motivating evaluation-first teaching rather than generation-first exercises (Meske et al., 2025; Sapkota et al., 2025).

2.2 Adoption and Effects

Adoption of large-language-model tools has risen quickly across U.S. knowledge work, with use concentrated in information-technology and other digital roles; nationally representative surveys show a clear climb from late 2024 into mid-2025, led by younger and more-educated workers (Hartley et al., 2025). Outside the United States, a representative Danish survey fielded November 2023–January 2024 reports very high awareness and substantial workplace use, with especially high uptake among software developers, higher use among younger and less-experienced workers, and meaningful constraints from employer restrictions and required training (Humlum & Vestergaard, 2024). Taken together, these surveys indicate that LLMs have already entered routine workflows for many developers, sharpening the stakes for entry-level readiness and for how curricula introduce AI-assisted practices.

Effects on productivity vary by context, task, and user experience. Randomized trials inside large firms report substantial increases in output and task completion when developers receive access to coding assistants, with the largest gains typically appearing among lower-tenure developers and on tasks aligned with current model strengths (Cui et al., 2025). In contrast, a randomized field experiment with experienced open-source contributors working on mature repositories found that allowing state-of-the-art assistants slowed completion, a result participants did not expect and that the authors link to verification costs and "almost-right" outputs that require additional wrangling (Becker et al., 2025). These studies indicate that benefits are real yet uneven across settings, and that verification demands can erode speed when tasks require careful integration or when users over-trust surface plausibility.

Evidence from classrooms shows conditional benefits. In a controlled study with novice programmers, sections permitted to use ChatGPT performed similarly to control sections, suggesting that instructors can allow tools under structured guardrails without harming core outcomes (Kosar et al., 2024). In contrast, a mixed-methods deployment study reports that unstructured tool use can degrade code quality and collaboration (most notably pair programming), indicating the need to revise learning objectives and interaction rules (Groothuijsen et al., 2024). At scale, providing timely, real-time AI feedback increases

engagement and yields substantive style improvements once basic functionality is in place, supporting low-latency formative checks in programming courses (Woodrow et al., 2024). Synthesizing these results, structured integration, rapid feedback, and explicit roles enable learning gains, while unstructured use risks collaboration and quality problems.

Labor-market indicators reinforce the educational implications. Using high-frequency U.S. payroll records through mid-2025, early-career workers in occupations most exposed to generative AI show relative employment declines, with adjustments appearing more in employment than in wages—patterns consistent with displacement pressures at the entry level (Brynjolfsson et al., 2025). In combination with survey evidence on rapid workplace adoption (Hartley et al., 2025) and mixed experimental results, the labor evidence supports a pragmatic claim: programs should prepare students to capture gains where they are likely and to mitigate verification costs where risks are salient (Becker et al., 2025; Brynjolfsson et al., 2025; Cui et al., 2025; Hartley et al., 2025).

2.3 Trust in Automation

Appropriate reliance on AI-assisted coding is a matter of calibration, aligning trust with system capability to avoid both misuse, where trust exceeds reliability, and disuse, where capable systems are underutilized. Classic automation studies established this framework by showing how trust interacts with situation awareness, workload, and complacency in shaping reliance (Lee & See, 2004; Parasuraman & Riley, 1997; Parasuraman et al., 2008). Trust in AI is not only an individual cognitive judgment but also arises from affective and social bases, as well as from features of the tool, task, and organizational setting, including transparency, reliability, and immediacy of interaction (Glikson & Woolley, 2020). Because generative systems produce outputs that may be fluent yet fallible, calibration requires evidence about reliability and process rather than reliance on fluency alone.

In coding assistants, developers infer perceived ability, benevolence, and integrity from cues such as suggestion quality, confidence or explanations of rationale, transparency about limitations, and user control; accordingly, calibrated trust is partly a design problem (Glikson & Woolley, 2020; Wang et al., 2024). Syntheses focused on software development report that trust updates with the comprehensibility and perceived correctness of suggestions, the risks and dependencies of the task, and the presence of evaluation and safety guardrails (Sabouri et al., 2025). These factors align with learning frameworks showing that deeper outcomes arise when students construct and interact rather than merely receive information (Biggs & Tang, 2011; Chi & Wylie, 2014). Together, they imply that interface affordances and classroom routines should prompt scrutiny, justification, and evaluation rather than passive acceptance.

Trust is also dynamic, evolving across performance episodes of setup, action, and reflection that parallel prompting, drafting, testing, and review in everyday development (McGrath et al., 2025). Critically, subjective trust and objective outcomes can diverge; users may feel more confident or efficient while accuracy or time costs do not improve, and verification and coordination costs can offset expected productivity gains, even as structured use and task alignment yield improvements (Becker et al., 2025; Cui et al., 2025; Qian & Wexler, 2024). Long-standing research on instructional practice shows that active engagement is most effective when it surfaces misconceptions and requires reasoning in real time, as in peer instruction implementations that produce substantial normalized learning gains in physics courses (Crouch & Mazur, 2001). These results underscore why calibration should rest on evidence and interactive routines rather than fluency alone.

Novices are especially prone to illusions of competence, accepting plausible but incorrect code; calibration improves when tasks make uncertainty visible and require justification (Chi & Wylie, 2014; Prather et al., 2024). Peer instruction models demonstrate that structured questioning and discussion can move students from initial misconceptions to durable understanding (Crouch & Mazur, 2001). Learning science research similarly shows that interactive and constructive engagement produces the deepest outcomes (Chi & Wylie, 2014). Calibration is further reinforced when tools deliver timely, in-editor feedback that prompts refinement after functionality is achieved, raising attention to style and quality (Woodrow et al., 2024). Accordingly, calibration can be taught through artifacts and routines, including tests that define "correct," review that interrogates diffs, and brief defenses that surface assumptions,

supplemented by interface affordances that provide confidence explanations, transparency about limitations, and evaluation and safety guardrails (Sabouri et al., 2025; Wang et al., 2024).

2.4 Pedagogical Integration

Instructor concerns about AI coding assistants center on overreliance, blurred boundaries between acceptable reuse and plagiarism, and assessment alignment (Sheard et al., 2024). Classroom studies report that students use chatbots for explanations and debugging, while instructors note declines in code quality and negative effects on pair programming and collaboration (Groothuijsen et al., 2024). Comparative evidence shows that AI-assisted pair programming yields comparable or better performance than individual work but lower perceived collaboration and social presence than human pairs (Fan et al., 2025). Together these findings suggest that pedagogical integration should be guided by structured policies and routines to protect learning value.

Common guardrails include requirements for disclosure and attribution and the use of process evidence, such as in-lab extensions and brief oral defenses of design choices, together with grading schemes that reward reasoning and verification as well as final code (Kosar et al., 2024; Sheard et al., 2024). Pedagogical code review provides a precedent for these practices, with structured roles, checklists, and rubric-ready artifacts that shift attention from output to reasoning (Hundhausen et al., 2009). Controlled studies indicate that such adjustments function as learning supports: redesigned assignments and oral defenses maintained outcomes when ChatGPT was permitted (Kosar et al., 2024), and AI-augmented instruction accelerated foundational learning while emphasizing balance with independent problem solving (McCulloh et al., 2025). Trust research likewise suggests that interface affordances such as confidence or rationale explanations, quality signals, and user control, can scaffold scrutiny and calibrated reliance in classroom contexts (Wang et al., 2024). These findings support evaluation-focused structures that neutralize risks while preserving benefits.

Evaluation-first pedagogy also requires scalable mechanisms for feedback that direct attention beyond initial functionality. Timely, in-editor interventions can prompt refinement once tests pass, shifting student focus to code quality and style when they are most receptive (Woodrow et al., 2024). Pair programming studies suggest that AI assistance can reduce opportunities for peer explanation relative to human pairs, which motivates adding scaffolds that prompt articulation during collaboration (Fan et al., 2025). Interface-level features can further reinforce this orientation by providing confidence or rationale explanations, quality signals, and user control in ways that align with classroom expectations of justification and defense (Wang et al., 2024). These strategies position AI not as a shortcut to working code but as a catalyst for evidence, critique, and evaluation, which are durable learning outcomes of programming education.

3. DATA AND METHODS

This study uses data from the Stack Overflow Developer Survey (SODS) for 2023–2025. SODS is an annual, voluntary, nonprobability survey administered by Stack Overflow to its user community. Each wave draws tens of thousands of respondents, predominantly professional developers and students in computing fields, though the survey is open to anyone who visits the site. Because SODS is industry-oriented rather than designed for educational research, it captures practicing developers' experiences rather than student or instructor perspectives directly. Following prior work that uses SODS to map role-specific technologies and curriculum-relevant skill signals, SODS is used here descriptively as an industry-facing indicator of trends relevant to curriculum design (Dada et al., 2023). The objective is to characterize patterns among survey respondents, not to produce population-representative estimates. Because SODS is a nonprobability sample with no defined sampling frame, inferential statistics (e.g., significance tests) are not appropriate; all findings are descriptive.

The following measures are used:

- *AI usage.* SODS asks, "Do you currently use AI tools in your development process?" Adoption is coded as a binary indicator equal to one for respondents reporting current use of any AI coding

assistant and zero otherwise. The adoption proportion is the mean of this indicator among respondents with nonmissing values.

- *Trust.* Trust in AI tool accuracy is standardized to a five-point scale, with 1 representing lowest trust and 5 highest trust, in response to "How much do you trust the accuracy of the output from AI tools as part of your development workflow?" Analyses report mean trust and the share classified as "high trust" (values of 4 or 5).
- *Experience.* Professional experience is measured using the work experience variable and binned into Junior (0–2 years), Mid-level (3–5), Senior (6–10), and Expert (10+). The same bands were applied in each year.
- *Frustrations (2025 only).* SODS asked, "When using AI tools, which of the following problems or frustrations have you encountered?" Multi-select responses are parsed on semicolons, trimmed, and deduplicated at the respondent–category level so that each respondent contributes at most once to each category. Because multiple selections are allowed, category shares can exceed 100 percent. The display reports the ten most common frustrations.
- *Learning channels (2025 only).* SODS asked, "How did you learn to code for AI in the past year?" Multi-select responses are parsed and standardized into the displayed categories: Online resources (blogs, tutorials, videos, forums), Documentation, Workplace training, Formal education, and Online courses. "Other" write-ins are not displayed. Each respondent is counted once per selected category.
- *Vibe coding (2025 only).* SODS included a free-text item that asked respondents, "In your own words, is 'vibe coding' part of your professional development work? For this question, we define vibe coding according to the Wikipedia definition, the process of generating software from LLM prompts." Responses were coded using keyword matching: text was lowercased and matched against affirmation patterns ("yes," "definitely," "absolutely," "part of"), negations ("no," "not," "never," "don't"), and hedges ("sometimes," "occasionally," "maybe"), producing four categories: *Yes*, *No*, *Sometimes*, and *Unclear*. Ambiguous responses (those matching none of these patterns or containing conflicting signals) were coded as *Unclear* and excluded from denominators. This approach is adequate for the study's descriptive aims: the goal is to estimate approximate prevalence among respondents who gave interpretable answers, not to achieve fine-grained classification. For displays, the vibe coding indicator equals one for *Yes* or *Sometimes* and zero for *No*.

Analyses use listwise deletion within each outcome. Respondents with missing values on the variables required for a given analysis are excluded from that calculation. No imputation is applied. Item availability varies across waves: adoption and trust are available in all three years; frustrations and learning channels are available only in 2025. As a result, temporal comparisons are restricted to measures present in each wave, while the 2025-only items are presented as cross-sectional descriptives. The usage-trust gap was computed as the adoption proportion minus the share reporting high trust.

4. RESULTS

From 2023 to 2025, adoption rises sharply while stated trust declines. The usage-trust gap widens each year as use grows faster than reports of high accuracy. Mean trust drifts downward over time, and by 2025 adoption reaches a clear majority among respondents. The direction is consistent on both measures (more use and lower trust), so the widening gap is the salient takeaway from Table 1.

Year	Sample Size	AI Usage Rate	Trust (1-5)	High Trust %	High Trust-Usage Gap
2023	89,184	44.4%	3.12	42.2%	2.2%
2024	65,437	61.8%	3.08	43.0%	18.8%
2025	49,123	78.5%	2.71	32.8%	45.7%

Table 1. AI Adoption and Trust Metrics Over Time, 2023–2025

Across 2023–2025, the usual pattern holds: juniors report slightly higher trust than experts, but the gaps shrink. By 2025, all experience bands cluster near 2.7 on the 1 to 5 scale, and the difference between juniors and experts is under a tenth of a point. Year over year, each band moves down by a similar amount, so the separation between bands narrows (Table 2).

Experience Level	2023		2024		2025	
	N	Mean Trust	N	Mean Trust	N	Mean Trust
Junior (0-2 years)	5,328	3.20	3,538	3.08	4,850	2.76
Mid-level (3-5 years)	8,609	3.14	5,961	3.05	7,334	2.77
Senior (6-10 years)	11,427	3.07	7,746	2.98	9,640	2.71
Expert (10+ years)	18,215	3.03	12,413	2.97	21,020	2.68

Table 2. Mean AI Trust by Professional Experience, 2023–2025

The most common frustrations fall into three groups. First, accuracy and reliability: many respondents report incorrect or "almost-right" outputs and the need to recheck results. Second, context and explanation: tools often miss project constraints, handle edge cases poorly, or provide thin or opaque rationales. Third, integration and verification overhead: time spent testing, revising, and fitting outputs into existing codebases and workflows. Smaller but meaningful shares note security and IP concerns and tool frictions such as inconsistency or instability. Figure 1 shows the rank order and shares.



Figure 1. Developer Frustrations with AI Tools

Figure 2 reports the distribution of learning channels for those who answered the learning item in 2025. Multi-select responses are counted once per respondent–channel, so totals exceed 100%. Most respondents report learning through *online resources* (73%), followed by *documentation* (42%) and *workplace training* (22%). Only small minorities name *formal education* (6%) or *online courses* (3%). These proportions indicate that skill formation for AI-assisted coding occurs primarily outside formal coursework, with workplace offerings playing a secondary role.

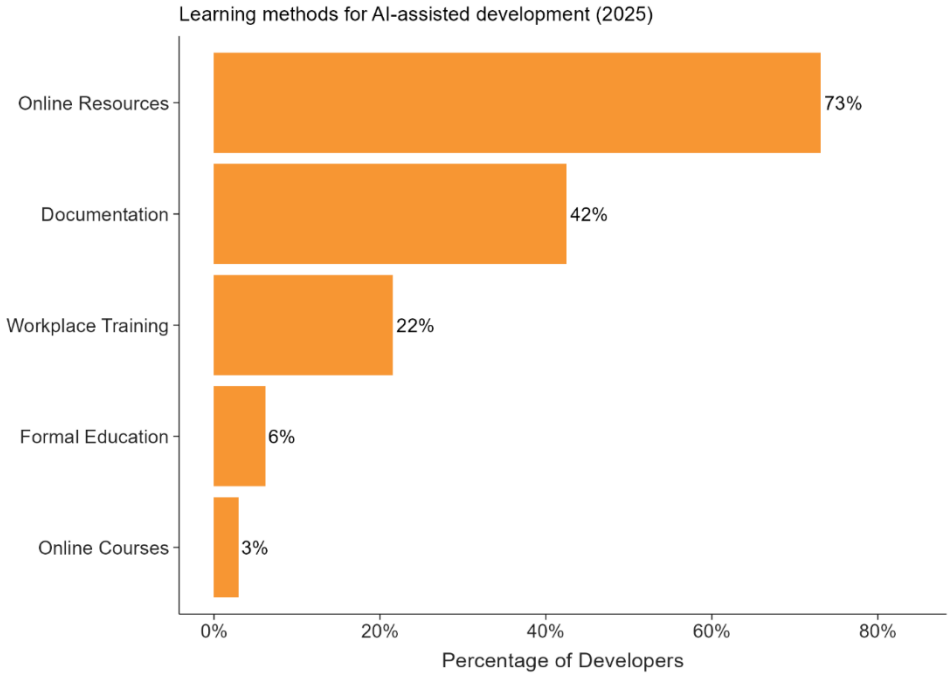


Figure 2. How Developers Learn AI Coding

Table 3 summarizes 2025 adoption of vibe coding. About one in five respondents report using it as part of their workflow. Adoption varies little by experience: junior, mid-level, senior, and expert groups cluster within a narrow range, with no clear upward or downward pattern. This indicates that vibe coding is broadly shared across seniority rather than concentrated among novices.

Experience Level	Vibe Coding %
Junior (0-2 years)	20.8
Mid-level (3-5 years)	20.9
Senior (6-10 years)	20.4
Expert (10+ years)	19.5

Table 3. Vibe Coding Adoption by Experience Level

5. DISCUSSION

The descriptive patterns reported above address each of the study's research questions and, together with the theoretical framework developed in Section 2, motivate the pedagogical implications developed in Section 6.

RQ1: Where do developers report learning AI-assisted coding? Learning is predominantly informal: online resources dominate, followed by documentation; workplace training is distant, and formal education

appears as a minority pathway (Figure 2). This distribution suggests an opportunity for formal coursework to provide structured evaluation routines that informal channels typically lack. Two strands of evidence align with this interpretation. First, timely AI feedback delivered in the editor scales post-functionality improvements in code style (Woodrow et al., 2024). Second, classroom studies report that unstructured chatbot use can reduce opportunities for peer explanation in pair programming unless collaboration is explicitly scaffolded (Groothuijsen et al., 2024). The finding also resonates with emerging accounts that tooling is shifting from syntax manipulation toward intent-to-implementation workflows, which lowers entry barriers while increasing the salience of evaluation depth (Gadde, 2025).

RQ2: What patterns in usage, trust, and experience characterize current AI coding tool adoption? Across 2023–2025, adoption rises while stated trust declines, yielding a steadily expanding usage-trust gap (Table 1). Interpreting this divergence through classic automation research points to a calibration problem: reliance can outpace or lag demonstrated capability, creating risks of misuse (over-reliance) and disuse (under-reliance) depending on task and context (Lee & See, 2004; Parasuraman & Riley, 1997). These results motivate designs and learning activities that make system purpose, process, and performance legible to users so that reliance tracks demonstrated capability (Lee & See, 2004; Wang et al., 2024).

The familiar ordering—juniors reporting higher trust than experts—persists across waves, but by 2025 the bands converge at approximately 2.7, leaving trust only weakly differentiated by seniority that year (Table 2). Interpreted through classic automation research, such compression is consistent with models in which trust is shaped chiefly by perceived reliability, situation awareness, and workload—factors that can cut across tenure—so reliance patterns need not track experience per se (Parasuraman et al., 2008). Instructional emphasis should therefore target calibrated use for all learners, not only novice-specific guardrails.

Respondents' top frustrations in 2025 cluster in three domains (Figure 1). First, accuracy and reliability concerns lead: incorrect or "almost-right" outputs that require additional checking. Second, context and explanation limits are common: difficulty incorporating project constraints and edge cases, and insufficient or opaque rationales. Third, integration and verification overhead is salient: time costs for testing, revising, and fitting outputs into existing codebases and workflows. Smaller segments flag security/IP issues and tool instability. These categories are consistent with well-documented risks of complacency and degraded monitoring under workload and time pressure in human-automation settings (Parasuraman et al., 2008), and they echo empirical findings that generative tools can amplify "illusions of competence" among novices without explicit metacognitive scaffolds (Prather et al., 2024).

RQ3: How prevalent is vibe coding among developers in 2025? Vibe coding appears in the 2025 cross-section as a broadly shared practice rather than one concentrated among novices: adoption is roughly similar across experience levels among respondents (Table 3). Conceptually, that breadth aligns with accounts that reposition expertise from typing every line to orchestrating and evaluating model contributions, including defining tests, reviewing changes, and integrating code under architectural and security constraints. This horizontal diffusion complements the usage-trust gap (Table 1) and the compressed trust gradient by 2025 (Table 2). These patterns support a calibration-for-all stance: if intent-mediated workflows are common across seniority, instruction should emphasize verification habits universally, not only for juniors.

The analysis presented here carries certain limitations. This study analyzed a large voluntary convenience sample from the Stack Overflow Developer Survey; all estimates are unweighted descriptives and should be interpreted as patterns among respondents rather than population parameters. Several methodological constraints warrant emphasis.

First, selection effects are plausible. Stack Overflow respondents are typically engaged community members and may differ from other developer populations. The respondent pool itself may have shifted compositionally across waves; for example, if higher-trust AI users increasingly rely on alternative forums or internal tooling and participate less in the Stack Overflow community by 2025, the observed decline in trust could partly reflect who is answering rather than a uniform change among developers writ large.

Second, SODS is an industry-oriented survey designed to capture trends among practicing developers, not students or educators. The findings therefore reflect workplace practices and may not generalize directly

to classroom settings. Triangulation with classroom-based data or qualitative evidence from instructors and students would strengthen confidence in the educational implications drawn here.

Third, temporal and causal scope are limited. The period 2023–2025 spans rapid changes in AI coding tools and developer practice; observed distributions may evolve as interfaces and norms shift. Cross-sectional descriptives cannot adjudicate whether learning channels, experience, or other factors cause differences in trust or reliance; unmeasured confounds and cohort effects remain plausible. Because SODS lacks longitudinal tracking of individual respondents, within-person changes in trust or practice cannot be observed.

Fourth, vibe coding remains an emerging and contested practice. The term itself was introduced in 2025, and professional bodies, employers, and educational institutions are still negotiating its boundaries, including questions of acceptable use, assessment standards, and the nature of expertise in AI-assisted work. Relatedly, ethical uncertainties surround reliance on generative models for software development, including concerns about code provenance, intellectual property, and accountability for errors. The pedagogical proposals in this paper should be understood as provisional responses to a rapidly evolving landscape, not as settled curricular standards.

Finally, the proposed teaching activities have not been empirically evaluated. Future work should pilot these designs in programming-intensive courses and assess effects on student calibration, learning outcomes, and attitudes toward AI-assisted coding.

These constraints argue for calibration-focused pedagogy that is robust to ongoing tool change rather than for strong claims about mechanism, and for treating the proposals in Section 6 as starting points for classroom experimentation.

6. PEDAGOGICAL IMPLICATIONS

The patterns presented across Sections 4 and 5 point toward pedagogical responses. The widening usage-trust gap and the convergence of trust across experience levels by 2025 indicate that verification habits should be explicit and universal, not limited to novice-specific guardrails. The frustration profile, dominated by accuracy concerns, explanation limits, and integration overhead, suggests that assessment should reward fault-finding, boundary checks, and the coordination of testing and documentation. The prevalence of vibe coding across experience levels reinforces the need to treat orchestration and evaluation as competencies in their own right. And the dominance of informal learning channels suggests an opportunity for formal coursework to supply the structured evaluation routines that self-directed learning typically lacks.

Synthesizing trust-in-automation research and active-learning frameworks with these findings offers four guiding principles. First, teach calibration as a routine practice for all learners: require evidence-based accept, revise, or reject decisions anchored in tests, invariants, and brief justifications (Lee & See, 2004; Parasuraman & Riley, 1997; Sabouri et al., 2025; Wang et al., 2024). Second, grade for evidence before volume: reward tests, short rationales, and reproducible processes using structures such as pedagogical code reviews and documented artifacts. Guardrails and oral defenses function as learning supports, not merely integrity checks (Hundhausen et al., 2009; Kosar et al., 2024; Sheard et al., 2024). Third, align tasks with common frictions by design: use "almost-right" debugging to target accuracy, specification-first work with rationale to target explanation, and integration diaries with lightweight automated checks to make verification overhead visible (Becker et al., 2025; Prather et al., 2024; Wang et al., 2024). Fourth, teach intent-first development under rapid change (Gadde, 2025). Pair timely, in-editor feedback that appears after student-owned tests pass with collaboration scaffolds that prompt articulation during pair work, so AI tools do not displace human explanation (Fan et al., 2025; Groothuijsen et al., 2024; Woodrow et al., 2024). These principles translate into four low-overhead activity types, described below. The Appendix provides sample templates for each activity.

6.1 Calibration Sprint: From "Almost-Right" to Verified

Instructors can operationalize calibrated reliance through short, structured cycles that directly target accuracy, reliability, and verification-overhead frictions (Becker et al., 2025). Students receive a compact

AI-generated solution seeded with realistic faults characteristic of almost-right outputs. They first author minimal failing tests to expose each defect, then revise the code, and conclude with a concise accept, revise, or reject decision justified by test evidence and a brief rationale (Lee & See, 2004; Parasuraman & Riley, 1997). Submissions include the failing tests, patched code, a before/after diff, a short justification, and an AI-use disclosure with process evidence where applicable (Sheard et al., 2024; Wang et al., 2024). Evaluation prioritizes fault coverage, test efficacy, and the quality of the warrant over code volume. This sequence operationalizes evidence-based decisions central to calibrated reliance and aims to mitigate trust–performance divergence in human–AI collaboration (Qian & Wexler, 2024).

Pedagogically, the sprint counters complacency and illusions of competence by positioning engagement at constructive or interactive levels via brief, structured peer-discussion checkpoints (Chi & Wylie, 2014; Crouch & Mazur, 2001; Prather et al., 2024). In trust-in-automation terms, the lab makes purpose, process, and performance auditable and leverages simple affordances—tests, transparency cues, and explicit decisions—to scaffold calibrated reliance (Lee & See, 2004; Sabouri et al., 2025; Wang et al., 2024).

6.2 Spec-to-Code With Constraints and Rationale

Instructors can frame a spec-to-code exercise that privileges explanation and constraints over volume: students identify the relevant passages from official documentation, name invariants and edge-case classes, author minimal tests that enforce those constraints, implement, and submit a bundle that includes a spec checklist with citations, an edge-case list, tests, code, a brief why-this-works rationale, an explicit accept, revise, or reject decision, and a one-line AI-use disclosure with process evidence where applicable. Assignment redesigns with short defenses have stabilized outcomes when AI is permitted, supporting the use of process evidence and justification as learning supports (Kosar et al., 2024). The rationale-and-test mapping moves work toward constructive or interactive engagement (Chi & Wylie, 2014; Crouch & Mazur, 2001), while trust research suggests that transparency and rationale explanations, user control, and evaluation guardrails are designable affordances that scaffold calibrated reliance (Lee & See, 2004; Parasuraman & Riley, 1997; Sabouri et al., 2025; Wang et al., 2024). After functionality is met, instructors can enable an in-editor style suggestion to prompt post-test refinement without displacing evaluation (Woodrow et al., 2024).

6.3 Integration Diary and Automated Verification

Instructors can make integration and verification overhead visible and therefore teachable by asking students to integrate an AI-generated snippet into an existing codebase while keeping a micro-diary of three moments: the first failing run, the first successful fix, and the final verification. Students record the task intent, any prompts used, the failing checks encountered, a concise before/after diff, and an explicit accept, revise, or reject decision with a one-sentence justification and an AI-use disclosure with process evidence when applicable. A small script can run unit tests, a style checker or type checker, and a basic dependency or security check so students triage failures, map defects to fixes, and document the evidence linking each defect to its remedy (Becker et al., 2025; Sabouri et al., 2025). Evaluation prioritizes the quality of error triage and evidentiary links over code volume, treating the diary and artifacts as process evidence rather than integrity policing (Kosar et al., 2024; Sheard et al., 2024). In team variants, instructors can assign light roles, such as navigator or explainer and verifier, to preserve peer explanation during AI-assisted work and reduce silent acceptance (Fan et al., 2025; Hundhausen et al., 2009). Framed this way, the diary operationalizes calibrated reliance in a setting where subjective confidence can diverge from objective outcomes, making purpose, process, and performance auditable (Qian & Wexler, 2024).

6.4 Lightweight Code Review for AI-Generated Code

This activity stages a short code review that centers evidence and explanation rather than polish. Instructors assign two roles (author and reviewer), time-box the exchange, and require a minimal packet that includes tests, a before/after diff, a brief rationale, and a one-line AI-use disclosure with process evidence where applicable. Reviewers identify specific risks (correctness, integration, security), point to the test or line

where the risk appears, and pose two focused questions. Authors respond with either a change or a defense, and both parties record an explicit accept, revise, or reject decision and the evidence that supports it.

This structure adapts classroom code-review precedents while motivating articulation during AI-assisted work (Fan et al., 2025; Hundhausen et al., 2009). Brief defenses and process artifacts function as learning supports, not just integrity checks, and have stabilized outcomes when AI is permitted (Kosar et al., 2024). By making transparency cues, constraints, and justification visible, the activity scaffolds calibrated reliance and helps guard against trust–performance divergence (Qian & Wexler, 2024; Wang et al., 2024). The interaction is intentionally interactive and constructive: students must question, justify, and revise, consistent with evidence that such engagement yields deeper learning than passive receipt (Chi & Wylie, 2014; Crouch & Mazur, 2001).

6.5 Implementation and Evaluation

Instructors can implement these activities with minimal overhead by standardizing a few templates and time boxes (Appendix). Sessions can be run in short cycles—for example, 20 to 30 minutes for work, 5 minutes for a peer check—to preserve interactive explanation (Chi & Wylie, 2014; Crouch & Mazur, 2001). Where available, enable simple, in-editor suggestions only after student tests pass; where not available, ask students to note one post-test refinement they made in response to feedback (Woodrow et al., 2024).

Evaluation should weight evidence and decisions at least as much as code: traceability from requirement to test to behavior, the clarity of the rationale, and the quality of the accept/revise/reject decision, alongside correctness and basic quality. Two small checks keep the effort empirical without formal experimentation: (1) a brief, per-lab calibration check where the decision is recorded with the specific evidence cited, and (2) a one-minute trust/performance pulse where students report confidence and number of tests passed, useful for spotting trust–performance divergence (Lee & See, 2004; Qian & Wexler, 2024). For teamwork, assign light roles (author/reviewer or navigator/verifier) to preserve peer explanation in AI-assisted settings (Fan et al., 2025; Hundhausen et al., 2009). Over time, review these artifacts in short episode debriefs to adjust prompts, constraints, or affordances, treating trust as dynamic and designable (McGrath et al., 2025; Sabouri et al., 2025; Wang et al., 2024).

7. CONCLUSION

This study situates the rapid shift toward AI-assisted coding within an information systems education perspective. Descriptive evidence from the 2023–2025 Stack Overflow Developer Survey reveals several patterns: adoption has climbed even as stated trust has softened; trust differences across experience levels persist but narrow by 2025; reported frustrations cluster around accuracy, explanation, and integration; learning occurs largely through informal channels; and vibe coding appears as a broadly shared practice rather than a novice phenomenon. These patterns support a curricular pivot from producing more code to reasoning more effectively in AI-assisted development, with evidence-first evaluation and orchestration as primary learning outcomes.

Trust-in-automation research provides conceptual footing for that pivot. Appropriate reliance is a matter of calibration, aligning trust with demonstrated capability to avoid misuse and disuse (Lee & See, 2004; Parasuraman & Riley, 1997). Because fluent but fallible outputs can elicit over- or under-reliance without explicit monitoring, structured verification remains necessary even when AI-generated code reads well (Parasuraman et al., 2008).

The pedagogical activities proposed here—calibration sprints, spec-to-code exercises, integration diaries, and structured code reviews—translate this theoretical stance into low-overhead classroom routines. Each activity foregrounds evidence-based accept, revise, or reject decisions; rewards verification over volume; and treats orchestration and evaluation as competencies in their own right. These proposals are intentionally pragmatic and are offered as starting points for pilot testing, not as settled standards.

Ultimately, as development shifts from line-by-line authorship toward AI-assisted coding, information systems education should lead by making calibrated reliance and reproducible process the visible standards of professional judgment. Courses that grade evidence over surface fluency, turn “almost-right” outputs

into verified solutions, and make integration costs explicit will prepare students to direct, audit, and improve AI systems rather than merely consume them. Done at scale, these moves do more than keep curricula current: they help establish the norms of responsible AI-assisted development, allowing educators rather than vendors to define the standards.

8. REFERENCES

Becker, J., Rush, N., Barnes, E., & Rein, D. (2025). Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity. *arXiv preprint arXiv:2507.09089*

Biggs, J., & Tang, C. (2011). Train-the-Trainers: Implementing Outcomes-Based Teaching and Learning in Malaysian Higher Education. *Malaysian Journal of Learning and Instruction*, 8, 1-19. https://doi.org/10.32890/mjli.8.2011.7624

Brynjolfsson, E., Chandar, B., & Chen, R. (2025). Canaries in the Coal Mine? Six Facts about the Recent Employment Effects of Artificial Intelligence. Working paper. *Latest version available at https://digitaleconomy.stanford.edu/publications/canaries-in-the-coal-mine*

Chi, M. T., & Wylie, R. (2014). The ICAP Framework: Linking Cognitive Engagement to Active Learning Outcomes. *Educational Psychologist*, 49(4), 219-243. https://doi.org/10.1080/00461520.2014.965823

Crouch, C. H., & Mazur, E. (2001). Peer Instruction: Ten Years of Experience and Results. *American Journal of Physics*, 69(9), 970-977. https://doi.org/10.1119/1.1374249

Cui, Z. K., Demirer, M., Jaffe, S., Musolff, L., Peng, S., & Salz, T. (2025). The Effects of Generative AI on High-Skilled Work: Evidence From Three Field Experiments With Software Developers. *Available at SSRN:* https://doi.org/10.2139/ssrn.4945566

Dada, O. A., Obaido, G., Sanusi, I. T., Aruleba, K., & Yunusa, A. A. (2023). Hidden Gold for IT Professionals, Educators, and Students: Insights From Stack Overflow Survey. *IEEE Transactions on Computational Social Systems*, 10(2), 795-806. https://doi.org/10.1109/TCSS.2022.3151130

Fan, G., Liu, D., Zhang, R., & Pan, L. (2025). The Impact of AI-Assisted Pair Programming on Student Motivation, Programming Anxiety, Collaborative Learning, and Programming Performance: A Comparative Study With Traditional Pair Programming and Individual Approaches. *International Journal of STEM Education*, 12(1), 16. https://doi.org/10.1186/s40594-025-00537-3

Fawzy, A., Tahir, A., & Blincoe, K. (2025). Vibe Coding in Practice: Motivations, Challenges, and a Future Outlook—A Grey Literature Review. *Proceedings of the IEEE/ACM 48th International Conference on Software Engineering: Software Engineering in Practice* (pp. 212-223). https://doi.org/10.1145/3786583.3786866

Gadde, A. (2025). Democratizing Software Engineering Through Generative AI and Vibe Coding: The Evolution of No-Code Development. *Journal of Computer Science and Technology Studies*, 7(4), 556-572. https://doi.org/10.32996/jcsts.2025.7.4.66

Gimpel, H., Hall, K., Decker, S., Eymann, T., Gutheil, N., Lämmermann, L., Braig, N., Maedche, A., Röglinger, M., Ruiner, C., Schoch, M., Schoop, M., Urbach, N., & Vandirk, S. (2025). Using Generative AI in Higher Education: A Guide for Instructors. *Journal of Information Systems Education*, 36(3), 237-256. https://doi.org/10.62273/QLLG7172

Glikson, E., & Woolley, A. W. (2020). Human Trust in Artificial Intelligence: Review of Empirical Research. *Academy of Management Annals*, 14(2), 627-660. https://doi.org/10.5465/annals.2018.0057

Groothuijsen, S., van den Beemt, A., Remmers, J. C., & van Meeuwen, L. W. (2024). AI Chatbots in Programming Education: Students' Use in a Scientific Computing Course and Consequences for Learning. *Computers and Education: Artificial Intelligence*, 7, 100290. https://doi.org/10.1016/j.caeai.2024.100290

Hartley, J., Jolevski, F., Melo, V., & Moore, B. (2025). Job Loss Fears in the First Years of Generative Artificial Intelligence. *SSRN*. https://doi.org/10.2139/ssrn.5136877

Humlum, A., & Vestergaard, E. (2024). The Adoption of ChatGPT. *University of Chicago, Becker Friedman Institute for Economics Working Paper* (2024-50). https://doi.org/10.2139/ssrn.4807516

Hundhausen, C., Agrawal, A., Fairbrother, D., & Trevisan, M. (2009). Integrating Pedagogical Code Reviews Into a CS 1 Course: An Empirical Study. *ACM SIGCSE Bulletin*, 41(1), 291-295. https://doi.org/10.1145/1539024.1508972

Kosar, T., Ostojić, D., Liu, Y. D., & Mernik, M. (2024). Computer Science Education in ChatGPT Era: Experiences From an Experiment in a Programming Course for Novice Programmers. *Mathematics*, 12(5), 629. https://doi.org/10.3390/math12050629

Lee, J. D., & See, K. A. (2004). Trust in Automation: Designing for Appropriate Reliance. *Human Factors*, 46(1), 50-80. https://doi.org/10.1518/hfes.46.1.50_30392

McCulloh, I., Rodriguez, P., Kumar, S., Gupta, M., Sharma, V. R., Johnson, B., & Johnson, A. N. (2025). Generative AI in Computer Science Education: Accelerating Python Learning With ChatGPT. *arXiv preprint arXiv:2505.20329*.

McGrath, M. J., Duenser, A., Lacey, J., & Paris, C. (2025). Collaborative Human-AI Trust (CHAI-T): A Process Framework for Active Management of Trust in Human-AI Collaboration. *Computers in Human Behavior: Artificial Humans*, 100200. https://doi.org/10.1016/j.chbah.2025.100200

Meske, C., Hermanns, T., von der Weiden, E., Loser, K. U., & Berger, T. (2025). Vibe Coding as a Reconfiguration of Intent Mediation in Software Development: Definition, Implications, and Research Agenda. *IEEE Access*, 13, 213242-213259. https://doi.org/10.2139/ssrn.5378201

Parasuraman, R., & Riley, V. (1997). Humans and Automation: Use, Misuse, Disuse, Abuse. *Human Factors*, 39(2), 230-253. https://doi.org/10.1518/001872097778543886

Parasuraman, R., Sheridan, T. B., & Wickens, C. D. (2008). Situation Awareness, Mental Workload, and Trust in Automation: Viable, Empirically Supported Cognitive Engineering Constructs. *Journal of Cognitive Engineering and Decision Making*, 2(2), 140-160. https://doi.org/10.1518/155534308X284417

Prather, J., Reeves, B. N., Leinonen, J., MacNeil, S., Randrianasolo, A. S., Becker, B. A., Kimmel, B., Wright, J., & Briggs, B. (2024, August). The Widening Gap: The Benefits and Harms of Generative AI for Novice Programmers. *Proceedings of the 2024 ACM Conference on International Computing Education Research* (Vol. 1, pp. 469-486). https://doi.org/10.1145/3632620.3671116

Qian, C., & Wexler, J. (2024, March). Take it, Leave it, or Fix it: Measuring Productivity and Trust in Human-AI Collaboration. *Proceedings of the 29th International Conference on Intelligent User Interfaces* (pp. 370-384). https://doi.org/10.1145/3640543.3645198

Ray, P. P. (2025). A Review on Vibe Coding: Fundamentals, State-of-the-Art, Challenges and Future Directions. *Authorea Preprints*. https://doi.org/10.36227/techrxiv.174681482.27435614/v1

Rose, K., Massey, V., Marshall, B., & Cardon, P. (2023). IS Professors' Perspectives on AI-Assisted Programming. *Issues in Information Systems*, 24(2), 178-190. https://doi.org/10.48009/2_iis_2023_115

Sabouri, S., Eibl, P., Zhou, X., Ziyadi, M., Medvidović, N., Lindemann, L., & Chattopadhyay, S. (2025). Trust Dynamics in AI-Assisted Development: Definitions, Factors, and Implications. *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)* (pp. 1678-1690). IEEE. https://doi.org/10.1109/ICSE55347.2025.00199

Sapkota, R., Roumeliotis, K. I., & Karkee, M. (2025). Vibe Coding vs. Agentic Coding: Fundamentals and Practical Implications of Agentic AI. *arXiv preprint arXiv:2505.19443*.

Sheard, J., Denny, P., Hellas, A., Leinonen, J., Malmi, L., & Simon. (2024, March). Instructor Perceptions of AI Code Generation Tools – A Multi-Institutional Interview Study. *Proceedings of the 55th ACM Technical Symposium on Computer Science Education* (Vol. 1, pp. 1223-1229). https://doi.org/10.1145/3626252.3630880

Wang, R., Cheng, R., Ford, D., & Zimmermann, T. (2024, June). Investigating and Designing for Trust in AI-Powered Code Generation Tools. *Proceedings of the 2024 ACM Conference on Fairness, Accountability, and Transparency* (pp. 1475-1493). https://doi.org/10.1145/3630106.3658984

Woodrow, J., Malik, A., & Piech, C. (2024, March). AI Teaches the Art of Elegant Coding: Timely, Fair, and Helpful Style Feedback in a Global Course. *Proceedings of the 55th ACM Technical Symposium on Computer Science Education* (Vol. 1, pp. 1442-1448). https://doi.org/10.1145/3626252.3630773

AUTHOR BIOGRAPHY



Jeremiah Bohr is an Assistant Professor of Information Systems in the School of Business at the University of Wisconsin Oshkosh. He completed his Ph.D. at the University of Illinois at Urbana-Champaign, and his research interests include computational research methods, unstructured data analysis, agentic coding, multi-agent systems, synthetic data generation, and AI governance. He has recently published articles in *AI & Society*, the *Journal of Innovation Management*, and *Climatic Change*. His teaching interests include application development, Python for data analysis, data visualization, and generative AI.

APPENDIX

Sample Activity Templates

Template 1: Calibration Sprint Worksheet

Time: 45-60 minutes (25-30 min individual, 10-15 min peer discussion, 5-10 min debrief)

Component	Description
Provided code	[Instructor inserts AI-generated solution with 2-3 seeded faults]
Task	Write minimal failing tests to expose each fault, then fix the code
Submission checklist	☐ Failing tests (before fixes) ☐ Patched code ☐ Before/after diff ☐ Accept/revise/reject decision with justification ☐ AI-use disclosure
Evaluation criteria	Fault coverage (40%), Test efficacy (30%), Rationale quality (30%)

Instructor Notes:

This activity targets accuracy and verification-overhead frustrations. Provide students with AI-generated code containing 2-3 realistic faults (e.g., off-by-one errors, missing edge-case handling, incorrect API usage). The goal is to develop evaluative judgment for identifying and fixing plausible-but-wrong outputs.

Template 2: Spec-to-Code Checklist

Time: 60-75 minutes (40-50 min individual, 15-20 min peer review of rationales, 5 min wrap-up)

Item	Complete	Notes
Spec citations from official documentation	☐	List specific sections/pages referenced
Named invariants (conditions that must always hold)	☐	e.g., "balance >= 0 after any transaction"
Edge-case classes identified	☐	e.g., empty input, maximum values, null
Minimal tests enforcing constraints	☐	At least one test per invariant/edge case
Implementation	☐	Code that passes all tests above
Why-this-works rationale (2-3 sentences)	☐	Explain how code satisfies spec constraints
Accept/revise/reject decision	☐	Based on test evidence
AI-use disclosure	☐	Tools used, extent of use, which portions

Instructor Notes:

This activity targets explanation and context frustrations by requiring students to ground implementation in explicit constraints before writing code. Students must cite documentation, identify invariants, and articulate why their solution satisfies the spec. The emphasis is on reasoning from requirements to implementation, not on code volume.

Template 3: Integration Micro-Diary

Time: 50-65 minutes (35-45 min integration work with diary entries, 10-15 min pair share, 5 min reflection)

Diary Entry	Student Records
Task intent	What are you trying to accomplish?
Prompts used (if any)	Exact prompts or "none"
First failing run	What failed? Which check (test, linter, type checker)? Exact error message?
Diagnosis	What do you think caused the failure?
First successful fix	What did you change? Why did you expect this to work?
Final verification	All checks passing? List evidence (test output, linter results)
Before/after diff	Attach or summarize key changes
Accept/revise/reject decision	Circle one + one-sentence justification citing specific evidence
AI-use disclosure	Tools used, extent of use

Instructor Notes:

This activity makes integration overhead visible by requiring students to document their debugging process as they incorporate AI-generated code into an existing codebase. The diary structure surfaces the verification work that practitioners report as a major frustration. Provide a small but realistic codebase and an AI-generated snippet that requires adaptation.

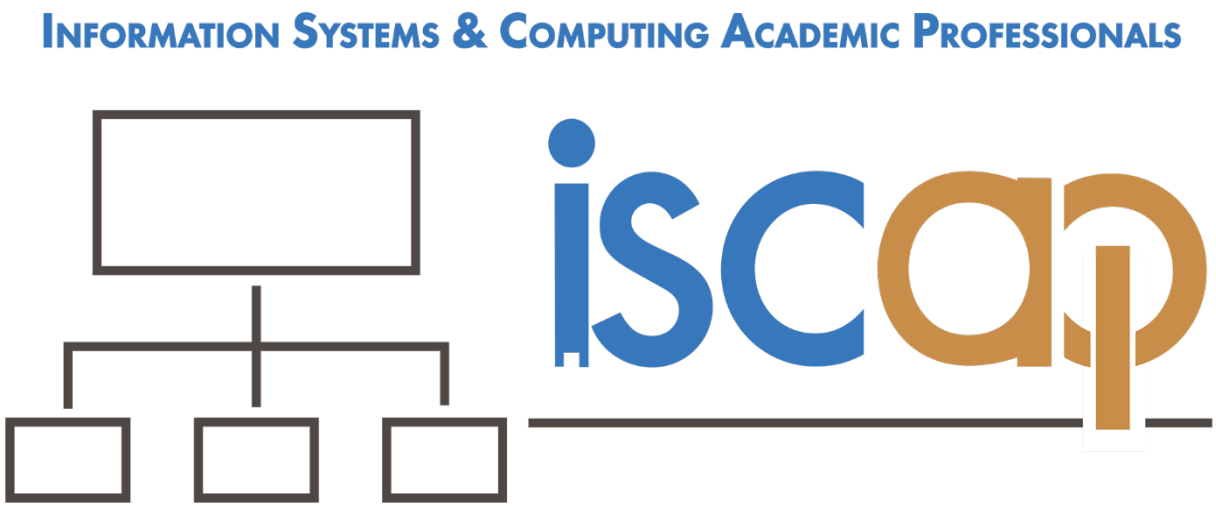
Template 4: Code Review Packet

Time: 40-50 minutes (10 min prep, 15-20 min review exchange, 10-15 min revision/response, 5 min debrief)

Item	Entry
Author Submits	
Tests covering key functionality	☐
Before/after diff	☐
Brief rationale (2-3 sentences)	☐
AI-use disclosure	☐
Reviewer Completes	
Risk 1: Type	☐ Correctness ☐ Integration ☐ Security
Risk 1: Location	Test # or Line #: ___
Risk 1: Question	[Focused question for author]
Risk 2: Type	☐ Correctness ☐ Integration ☐ Security
Risk 2: Location	Test # or Line #: ___
Risk 2: Question	[Focused question for author]
Author Responds	
Risk 1 response	☐ Change made: ___ ☐ Defense: ___
Risk 2 response	☐ Change made: ___ ☐ Defense: ___
Both Record	
Final decision + supporting evidence	☐ Accept ☐ Revise ☐ Reject Evidence: ___

Instructor Notes:

This activity structures peer review to center on evidence and justification rather than surface polish. Assign author and reviewer roles; the reviewer must identify specific risks and pose focused questions, and the author must respond with either a change or a defense. Both parties record explicit decisions.



STATEMENT OF PEER REVIEW INTEGRITY

All papers published in the ***Journal of Information Systems Education*** have undergone rigorous peer review. This includes an initial editor screening and double-blind refereeing by three or more expert referees.

Copyright ©2026 by the Information Systems & Computing Academic Professionals, Inc. (ISCAP). Permission to make digital or hard copies of all or part of this journal for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial use. All copies must bear this notice and full citation. Permission from the Editor is required to post to servers, redistribute to lists, or utilize in a for-profit or commercial use. Permission requests should be sent to the Editor-in-Chief, ***Journal of Information Systems Education***, editor@jise.org.

ISSN: 2574-3872 (Online) 1055-3096 (Print)