2026 ASEE ANNUAL CONFERENCE June 21–24, 2026 | Charlotte Convention Center, NC
Host Sponsor UNC Charlotte
ASEE AMERICAN SOCIETY FOR ENGINEERING EDUCATION
Paper ID #53823

# Work-in-Progress: Investigating Novice End-User Programmers' Problem-Solving Through Student–GenAI Interaction Patterns

**Mrs. Fadhla Binti Junus, Purdue Engineering Education**

A former Assistant Professor and Tenured Lecturer in Information Technology at the Department of Science and Technology, State Islamic University Ar-Raniry, Banda Aceh, Indonesia. Currently pursuing a Ph.D. at the School of Engineering Education, Purdue University, Indiana, USA. Her research uniquely combines industry experience with academic expertise, focusing on technology-enhanced learning. Specifically, her work centers around developing personalized learning environments for higher education students studying computer programming. She is particularly interested in investigating students' programming learning processes, exploring methods to simplify programming instruction, examining theoretical foundations for effective instructional design, and integrating artificial intelligence technologies to facilitate peer-like knowledge construction.

**Dr. Sean P Brophy, Purdue University – West Lafayette (College of Engineering)**

Dr. Sean Brophy is a learning scientist and mechanical engineer who develops, and research engineering learning environments enhanced with technology. His work spans multiple disciplines including aerospace, biomedical, civil, electrical, and mechanical engineering. His focus has been on the use of physical computing to enhance students' design and computational thinking. His latest research involves the integration of artificial intelligence for improving pedagogy, assessment, and learning goals in engineering learning environments.

# Work-in-Progress: Investigating Novice End-User Programmers' Problem-Solving Through Student–GenAI Interaction Patterns

This work-in-progress report presents findings from a pilot study examining how novice end-user programmers solve problems through interactions with their preferred generative artificial intelligence (GenAI) platform in a first-year engineering (FYE) course. We introduce the term "novice end-user programmers" (abbreviated as NEUPs) to describe learners at the early stages of acquiring programming skills to address problems within their specific domains. This term extends the concept of end-user programmers described in the literature as non-computer science (CS) individuals who engage in programming activities but do not aim to become professional programmers or software developers. FYE students are a typical example of NEUPs, as they participate in computer programming education designed to teach them to solve engineering problems using specific programming languages. As GenAI tools increasingly facilitate learning in programming education, understanding how students communicate, reason, and develop solutions during GenAI-supported problem-solving is highly important. Guided by our student–GenAI interaction patterns framework, this study analyzes students' prompt logs from their GenAI chat histories as they completed an in-class programming assignment. While quantitative content analysis was used to measure the descriptive distribution of students' prompt-engineering techniques, qualitative content analysis was employed to identify interaction patterns and behaviors. Findings suggest various characteristics of students' interactions with GenAI, including iterative feedback on GenAI's output and varied interaction patterns and behaviors. This ongoing work offers insights into improving our instructional design and the subsequent phases of data collection and analysis in our larger project, which is expected to inform educators and decision-makers at the college level in managing GenAI integration into engineering educational environments.

Keywords: generative AI, problem-solving, novice end-user programmers, chat log analysis, prompt engineering, first-year engineering education.

## Introduction

GenAI tools have demonstrated their potential to provide personalized, comprehensive, and immediate feedback. Prior work shows that such feedback supports students' learning across technical and conceptual programming tasks, from code generation and debugging to core fundamentals such as algorithmic problem-solving [1], [2], [3], [4]. However, these studies focus on novice programmers enrolled in introductory computer science (CS) programming courses (e.g., CS1 and CS2) who aspire to become professional programmers or software developers. This leaves a gap in understanding how GenAI can support novice end-user programmers (NEUPs).

NEUPs are individuals who engage in programming tasks to address domain-specific problems rather than pursuing careers as professional software developers. Examples include first-year engineering (FYE) students who use programming languages like Python and MATLAB to solve engineering problems. They face similar cognitive and technical challenges to their CS counterparts. Cognitive challenges cover the abstract nature of programming, syntax and semantics, as well as algorithm design, while technical difficulties include debugging and error

diagnosis [5], [6], [7], [8], [9], [10]. Beyond these shared challenges, NEUPs often lack formal CS training, leading them to rely on modifying existing code in ways that degrade code quality and erode their confidence in identifying themselves as programmers [11], [12]. To address these challenges, Myers et al. [5] suggested using a form of *programming by demonstration* (PBD). It is a system that often incorporates AI techniques to generalize user examples into reusable programs. They contended that PBD can make programming learning easier for NEUPs by providing structured guidance that minimizes complexity during programming tasks. GenAI tools can serve as a PBD system by providing accessible, on-demand assistance. Therefore, integrating GenAI into formal learning environments may help FYE students mitigate cognitive and technical programming difficulties.

This work-in-progress (WIP) paper reports findings from a pilot investigation of how novice end-user programmers interact with GenAI tools during problem-solving activities in a FYE course. Specifically, our pilot investigation aims to characterize interaction patterns that emerge in students' prompt logs as they work through a programming assignment to solve a complex problem (requiring multiple logic constructs within an algorithm). In addition, this pilot study addresses a gap in the literature that has predominantly focused on GenAI use in CS-related fields. Ultimately, this pilot study provides insights into our larger, ongoing research, improving our instructional design and subsequent phases of data collection and analysis.

**Conceptual Framework**

We propose a conceptual framework to characterize novice end-user programmers' interaction patterns with GenAI. We ground our framework in empirical studies from CS contexts [13], [14], [15], [16], recognizing that novice end-user programmers face many of the same programming challenges as novice programmers, and given the limited evidence on how GenAI supports programming learning in first-year engineering contexts. This framework is a mid-level component of our broader conceptual framework on GenAI's peer-like roles and scaffolding mechanisms in GenAI-mediated programming learning. In this WIP paper, we refer to this mid-level component as the **student-GenAI interaction patterns** framework because it foregrounds students' prompting, refinement, and evaluation of GenAI responses during help-seeking in programming. Although we applied the framework deductively as an initial analytic lens, we also documented any emergent patterns that did not fit within the framework to inform its future refinement in our larger project. Fig. 1 illustrates the framework, which delineates three levels of interaction patterns with GenAI: low, moderate, and high. In our larger project, these interaction patterns are interpreted alongside our theoretical framework of GenAI-Mediated Learning, which is informed by Social Constructivism [17], Cognitive Load Theory [18], Self-Regulated Learning [19], and Human-AI Interaction Guidelines [20].

We define each interaction level shown in Fig. 1 as follows. The low-level interaction refers to a pattern of minimal engagement in which students use GenAI passively and superficially, treating it as an answer provider that helps them find quick solutions to their problems. This interaction level is characterized by writing short prompts and copying and pasting task descriptions from instructors. In addition, students often accept GenAI's responses without further follow-up to verify their accuracy, relevance, or alignment with the task requirements. These characteristics reflect limited cognitive engagement or problem-solving effort, which can result in shallow

learning and a limited understanding of programming concepts, and may lead students to overreliance on GenAI.

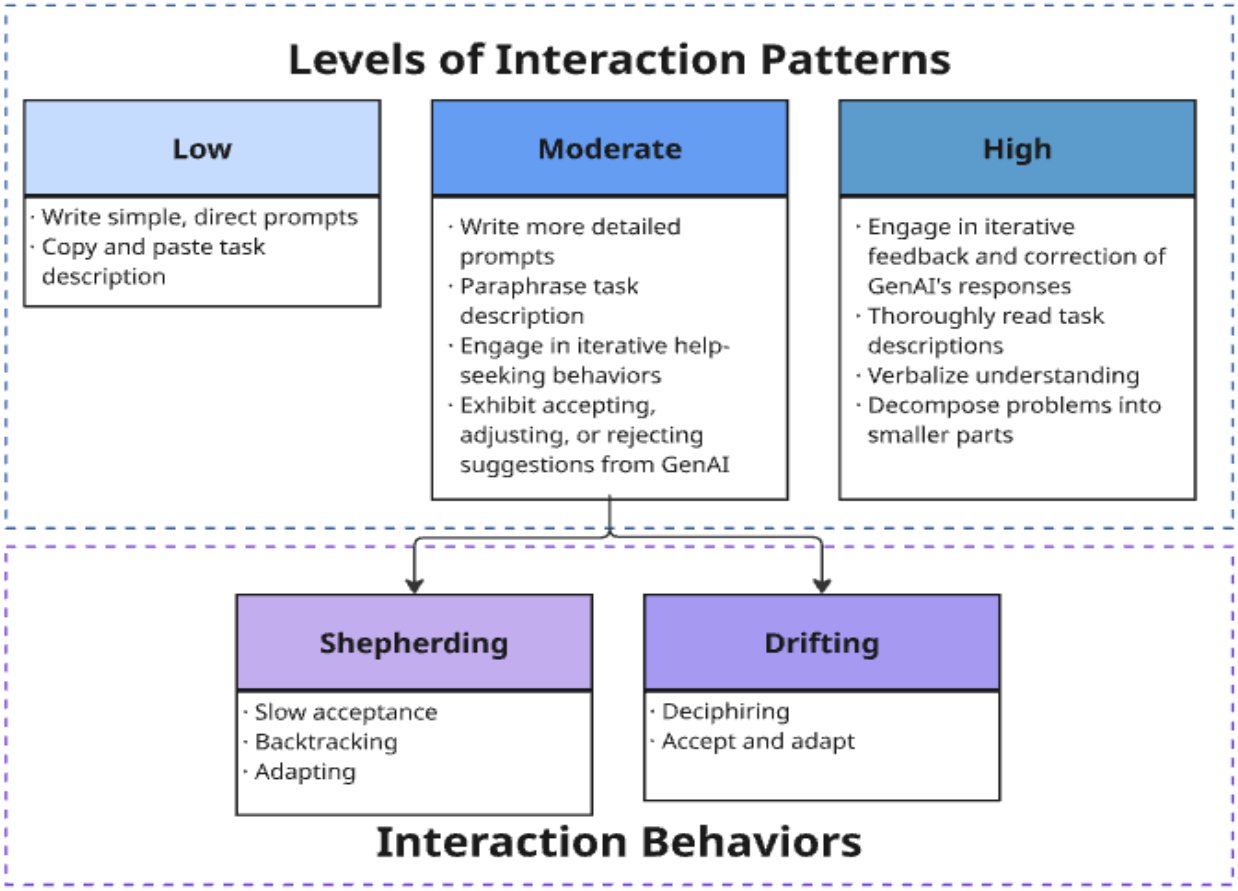


Fig. 1. Students-GenAI Interaction in Programming Learning.

Then, at the moderate level, students maintain a balanced level of engagement with GenAI by writing more detailed prompts, paraphrasing the assignment instructions, and engaging in iterative help-seeking behaviors (e.g., clarifying GenAI's answers with other sources or alternative GenAI platforms). This level reflects a procedural approach to learning, in which students apply trial and error to refine their understanding and solutions. We further classified this interaction level into two behaviors, as defined by Prather et al. [15], namely *shepherding* and *drifting.* The former refers to iterative attempts in guiding GenAI to generate the intended output. Students exhibit three characteristics: (a) *slow accept*, where they carefully review GenAI's suggestions, typing characters one by one or making incremental changes rather than accepting the output immediately; (b) *backtracking*, which is characterized by students accepting GenAI's suggestions but then deleting or modifying them upon realizing they are incorrect or unhelpful; (c) *adapting* is characterized when students accept GenAI's suggestions and actively modify them to better align with their needs or task requirements. In contrast, the latter behavior, *drifting*, reflects a repeated process of adapting the generated suggestions throughout the solution development process. There are two characteristics of *drifting*: first, *deciphering*, which characterizes students spending time understanding unhelpful or incorrect GenAI responses; and second, *accept and adapt,* in which students repeatedly accept, modify, and delete GenAI

suggestions through cycles of trial-and-error revisions without fully understanding the underlying logic.

Finally, the high level of interaction reflects deep, strategic engagement, in which students write detailed, context-rich prompts as they verbalize their thoughts after thoroughly reading the task descriptions, mirroring professional programming workflows. Students also engage in iterative prompt writing to critically evaluate GenAI's outputs. Additionally, they outline their problem-solving approaches (e.g., algorithms, pseudocode) and the existing code when requesting further explanation of the underlying logic. Such interaction patterns help students build robust programming skills, critical thinking, and computational literacy, while gaining confidence to work independently with GenAI as a supportive tool to enhance their productivity in developing solutions.

**Methods**

We collected data for our pilot study in Fall 2025 from a first-year engineering course comprising 90 students at a Midwestern public university. We asked students to use their favorite GenAI tools to solve a programming problem using MicroPython for a micro:bit (nRF52 processor). To prevent students from copying the assignment description, the instructor provided the problem instructions verbally. Students then interacted with GenAI tools and submitted links to their chats via Gradescope at the end of the class session.

We retrieved 78 submissions containing URL links to ChatGPT, Claude, Copilot, Gemini, and Purdue GenAI Studio. After cleaning the data and removing duplicate submissions, we obtained 61 submissions. We then accessed each link, saved the accessible chats by submission ID, and documented their accessibility status. This process yielded 41 eligible prompt logs after excluding 16 inaccessible links and four submissions without usable prompts.

We then conducted quantitative and qualitative content analyses following the guidelines of Krippendorff [21] and Schreier [22]. Prompt logs were first segmented into turn-level units. For quantitative content analysis, we treated each user turn (student-written prompt) as the unit of analysis. Within each submission ID, we counted the number of student prompts. We then computed prompt length as the average word count per prompt to normalize for differences in the number of prompts students wrote. Subsequently, we quantified the number of iterative prompts (i.e., prompts − 1, minimum 0). Finally, we summarized these submission-level metrics across the dataset to report overall averages.

For qualitative content analysis, prompt logs were analyzed deductively using the student-GenAI interaction patterns framework. Each interaction episode in the prompt log was then coded as low, moderate, or high based on observable features specified in the framework (e.g., minimal versus elaborated task context, evidence of iterative refinement, and the extent to which prompts reflected strategic engagement during problem solving). When a prompt was ambiguous, we checked adjacent turns to understand its context before coding. Additionally, for each moderate level of interaction pattern, we further coded the unit of analysis into *shepherding* or *drifting*. *Shepherding* reflected instances in which students actively steered the interaction toward convergence (e.g., backtracking, adapting constraints, and revising prompts after evaluating prior

outputs), whereas *drifting* reflected accept-and-adapt cycles with comparatively limited evidence of critique or convergence-oriented checks in the prompt text. Ultimately, we quantified the frequency and percentage distributions of coded interaction patterns and behaviors.

The analysis in this WIP paper was primarily deductive and focused on applying the student-GenAI interaction patterns framework. Although we also documented potential patterns not captured by the framework codes, this WIP paper reports only the deductive codes aligned with the conceptual framework. These codes directly support the study's focus on interaction patterns and behaviors.

## Results

### Quantitative Findings

A total of 41 accessible links were included for final analysis. Table I summarizes the quantification of prompt log datasets and students' interactions with GenAI. For prompt logs, it reports the average number of prompts per submission (6.95), average word count per prompt (27.56), and average number of iterative prompts (5.95). As for interaction patterns, the distribution of prompts across interaction levels shows that most students exhibited a moderate level of interaction, with relatively few high-level prompts. Lastly, more students performed *drifting* behaviors when interacting with GenAI.

**TABLE I**
**PROMPT LOG METRICS AND CODED INTERACTION PATTERNS AND BEHAVIORS**

| | | | | | |
|---|---|---|---|---|---|
| **Average Prompts Per Submission** | 6.95 | **Percentage of Low-Level Interaction** | 40.7 | **Percentage of *Shepherding*** | 31.7 |
| **Average Words Per Prompts** | 27.56 | **Percentage of Moderate-Level Interaction** | 53.7 | **Percentage of *Drifting*** | 68.3 |
| **Average Iterative Prompts** | 5.95 | **Percentage of High-Level Interaction** | 5.6 | | |

### Qualitative Findings

To contextualize the quantitative distributions reported in Table I, we classified the coding scheme for both interaction patterns and behaviors, as described in Tables II and III. Table II lists the operational criteria used to classify students' interaction patterns based on their prompt texts. For low-level interaction, prompts were typically short and straightforward, often asking for an immediate solution or explanation without providing context or expected outputs. Prompt patterns indicating moderate-level interaction often included additional context and clarification but lacked evaluation or iteration. Finally, high-level prompts elicited more active interaction, including iterative refinement, rejection or revision of prior suggestions, breaking the problem down into smaller steps, and verbalizing understanding.

**TABLE II**
**INTERACTION PATTERNS CODING SCHEME**

| Interaction Level | Operational Indicators | Typical Prompt Purposes |
|---|---|---|
| Low | • Simple/direct request (minimal context)<br>• Copy/paste task prompt without steering<br>• Few/no constraints (inputs/outputs/tests)<br>• No critique or revision language | • Quick solution for fixing code and error meaning<br>• Direct code generation request<br>• One-shot explanation |
| Moderate | • Adds some task context or constraints<br>• Paraphrases requirements<br>• Clarifying question(s)<br>• Limited evidence of evaluation/iteration | • Constraint-based generation<br>• Debugging with partial context<br>• Explanation + example request |
| High | • Iterative refinement (e.g., "I tried…, still…")<br>• Critique/rejection of output ("not quite…")<br>• Backtracking ("revert…," "previous version…")<br>• Decomposition/stepwise planning<br>• Verbalized understanding ("I think… so…") | • Debugging with testing/verification intent<br>• Decomposed problem solving<br>• Comparing alternatives / checking correctness |

Table III contrasts two interaction behaviors—*shepherding* versus *drifting*. It illustrates how students directed, revised, and verified GenAI outputs. S*hepherding* sessions were characterized by students directing over multiple turns, such as going back when their outputs did not satisfy the instruction requirements, changing the constraints or inputs, and using language that suggested evaluation. In contrast, *drifting* sessions featured recurrent accept-and-adapt cycles with fewer critique statements. Additionally, students frequently requested further modifications without explicit verification or a statement of why a prior output was insufficient.

## Discussion

The results indicate novice end-user programmers (i.e., first-year engineering students in this study) interact with GenAI at varying levels of autonomy and evaluative control. While most students' prompts included some context or constraints, the behavior sessions indicate that such information does not necessarily translate into convergence-oriented problem-solving. *Drifting* behavior often reflects repeated accept-and-adapt cycles with limited language for critique or verification. The following excerpt exemplifies *drifting* behavior in a student's sequence prompt

log, in which the interaction progresses by adding new requests rather than diagnosing mismatches or checking requirements.

Initial prompt: “Generate Python tilt sensor code.”
Follow-up prompt: “Now generate Python code that can detect tilt …”
Subsequent follow-up prompt: “Now generate a code that mimics something like …”

**TABLE III**
**INTERACTION BEHAVIORS CODING SCHEME**

| Interaction Behaviors Sessions | Behavioral Moves | Prompt Language |
|---|---|---|
| *Shepherding* | • Slow acceptance (conditional/qualified uptake)<br>• Backtracking after a mismatch<br>• Adapting constraints/inputs<br>• Convergence checks (verify/test/confirm) | • “I tested…, still fails…”<br>• “That doesn’t meet X requirement…”<br>• “Let’s go back and…”<br>• “Can you verify/check…” |
| *Drifting* | • Accept-and-adapt cycles<br>• Limited critique language<br>• Few/no explicit verification steps<br>• Iteration without convergence markers | • “Ok, now do …”<br>• “Modify this to…”<br>• Minimal “why/verify/test” cues |

In contrast, *shepherding* behavior showed more strategic monitoring, as students reported outcomes, backtracked when solutions violated constraints, and refined prompts based on results. For example, one student moved from an initial request, “Create a code script that will make a …,” to a refinement prompt, “Add so that it will do diagonal arrows as well,” and then to a backtracking prompt, “Ok, so let’s revert back to the below code because ….” This sequence reflects active steering toward convergence, in which the student extends requirements and then explicitly backtracks (“let’s revert back…”) to correct course. That backtracking/revision move is a concrete indicator of strategic monitoring rather than simple accept-and-adapt use.

Taken together, findings from the interaction patterns and behaviors highlight a potential misalignment between students’ immediate goals (e.g., obtaining error-free code quickly) and the disciplinary practices of programming, such as testing, error interpretation, and requirement validation. This suggests that the primary distinguishing factor may not be prompt length or specificity alone, but whether students externalize evaluative moves, such as error interpretation and requirement checking, within the GenAI dialogue. This distinction is also evident across interaction levels as shown in the excerpts below.

- Low interaction level: “Generate Python tilt sensor code.”

- Moderate interaction level: “For a 5x5 micro:bit display, make a code that …” (adds context/specifications)
- High interaction level: “Ok, so let’s revert back to the below code because …” (explicit backtracking/revision)

These results motivate instructional supports that make verification and reflection apparent in the interaction. Also, the results provide an empirical basis for refining the proposed conceptual framework of how agency, critique, and convergence behaviors shape GenAI-mediated problem-solving in first-year engineering programming contexts.

## Conclusion

In this work-in-progress study, we explored how novice end-user programmers (NEUPs) interact with generative artificial intelligence (GenAI) tools to solve programming problems in a first-year engineering (FYE) course. Guided by our conceptual lens of student-GenAI interaction in programming learning, we analyzed students’ prompt-log submissions to identify interaction patterns (low/moderate/high) and, at the moderate level, interaction behaviors (i.e., *shepherding* and *drifting*). Our findings revealed that most students engaged in moderate-level interactions, with drifting being the predominant behavior. This suggests that while students use GenAI tools to address programming challenges, their interactions often lack evaluative moves such as verification and critique, which are essential for effective programming problem-solving.

These insights highlight the need for instructional strategies that emphasize critical thinking, iterative refinement, and verification in GenAI-mediated problem-solving. By fostering these skills, educators can help NEUPs develop a deeper understanding of programming concepts and improve their confidence as programmers. Furthermore, the findings provide a foundation for refining the instructional intervention design and further data collection and analysis of our larger project, which can guide future research and inform the integration of GenAI tools into engineering education. Future work will focus on expanding the dataset, exploring additional interaction patterns using a hybrid coding approach, and investigating the impact of targeted instructional interventions on students’ learning outcomes and experiences.

## Acknowledgement

The authors acknowledge the support of Purdue’s College of Engineering Dean’s Office for providing resources that enabled the exploration of innovative uses of artificial intelligence in a first-year engineering course. We also thank the instructional staff and students whose engagement and feedback contributed directly to this work.

## References

[1] O. Hazzan and Y. Erez, “Rethinking Computer Science Education in the Age of Genai,” 2024. doi: 10.2139/ssrn.4992837.

[2] M. Kazemitabaar *et al.*, “CodeAid: Evaluating a Classroom Deployment of an LLM-based Programming Assistant that Balances Student and Educator Needs,” in *Conference on Human Factors in Computing Systems - Proceedings*, Association for Computing Machinery, May 2024. doi: 10.1145/3613904.3642773.

[3] T. H. Feng, A. Luxton-Reilly, B. C. Wünsche, and P. Denny, “From Automation to Cognition: Redefining the Roles of Educators and Generative AI in Computing Education,” Dec. 2024, [Online]. Available: http://arxiv.org/abs/2412.11419
[4] C. Lee, J. Myung, J. Han, J. Jin, and A. Oh, “Learning from Teaching Assistants to Program with Subgoals: Exploring the Potential for AI Teaching Assistants,” Sep. 2023, [Online]. Available: http://arxiv.org/abs/2309.10419
[5] B. A. Myers, A. J. Ko, and M. M. Burnett, “Invited research overview: End-user programming,” in *CHI ’06 extended abstracts on Human factors in computing systems*, Montréal, Québec, Apr. 2006, pp. 75–80.
[6] A. Robins, J. Rountree, and N. Rountree, “Learning and teaching programming: A review and discussion,” *Computer Science Education*, vol. 13, no. 2, pp. 137–172, 2003, doi: 10.1076/csed.13.2.137.14200.
[7] A. V. Robins, *Novice Programmers and Introductory Programming*. 2019. doi: 10.1017/9781108654555.013.
[8] M. Guzdial and B. du Boulay, “The history of computing education research,” in *The Cambridge Handbook of Computing Education Research*, 2019, pp. 11–39. doi: 10.1017/9781108654555.002.
[9] E. A. Youngs, “Human Errors in Programming,” *Int. J. Man. Mach. Stud.*, vol. 6, no. 3, pp. 361–376, May 1974, doi: 10.1016/S0020-7373(74)80027-1.
[10] D. McCall and M. Kölling, “A new look at novice programmer errors,” *ACM Transactions on Computing Education*, vol. 19, no. 4, 2019, doi: 10.1145/3335814.
[11] M. Guzdial, “Computing for other disciplines,” *The Cambridge Handbook of Computing Education Research*, pp. 584–605, 2019, doi: 10.1017/9781108654555.020.
[12] C. Scaffidi, M. Shaw, and B. Myers, “Estimating the numbers of end users and end user programmers,” in *Proceedings of the 2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC’05)*, 2005.
[13] N. Hashmi, Z. Li, S. Parise, and G. Shankaranarayanan, “Generative AI’s impact on programming students: Frustration and confidence across learning styles,” *Issues In Information Systems*, vol. 25, no. 3, pp. 371–385, 2024, doi: 10.48009/3_iis_2024_128.
[14] A. Scholl, D. Schiffner, and N. Kiesler, “Analyzing Chat Protocols of Novice Programmers Solving Introductory Programming Tasks with ChatGPT,” *arXiv preprint arXiv:2405.19132.*, May 2024, [Online]. Available: http://arxiv.org/abs/2405.19132
[15] J. Prather *et al.*, “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 - Volume 1, ICER 2024, Melbourne, VIC, Australia, August 13-15, 2024*, P. Denny, L. Porter, M. Hamilton, and B. B. Morrison, Eds., ACM, 2024, pp. 469–486. doi: 10.1145/3632620.3671116.
[16] R. Choudhuri *et al.*, “Insights from the Frontline: GenAI Utilization Among Software Engineering Students,” Dec. 2024, [Online]. Available: http://arxiv.org/abs/2412.15624
[17] L. S. Vygotsky, *Mind in society*. Harvard University Press, 1978.
[18] J. Sweller, “Cognitive Load During Problem Solving: Effects on Learning,” *Cogn. Sci.*, vol. 12, no. 2, pp. 257–285, Apr. 1988, doi: 10.1207/s15516709cog1202_4.
[19] B. J. Zimmerman, “Becoming a Self-Regulated Learner: Which Are the Key Subprocesses?,” 1986.

[20] S. Amershi *et al.*, “Guidelines for human-AI interaction,” in *Conference on Human Factors in Computing Systems - Proceedings*, Association for Computing Machinery, May 2019. doi: 10.1145/3290605.3300233.
[21] Klaus. Krippendorff, *Content analysis: An introduction to its methodology*. Sage, 2004.
[22] M. Schreier, *Qualitative content analysis in practice*. Sage, 2012.